Generación de historias centradas en los personajes mediante simulaciones

Trabajo de Fin de Grado

Juan Orellana Quemada
Samuel Rodríguez Oliva
Oscar David Vásquez Peraza

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2016

Documento maquetado con TEXIS v.1.0+.

Generación de historias centradas en los personajes mediante simulaciones

Trabajo de Fin de Grado

Dirigida por el Doctor

Gonzalo Méndez Pozo
Raquel Hervás Ballesteros

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2016

AUTORIZACIÓN PARA LA DIFUSIÓN DEL TRABAJO FIN DE GRADO Y SU DEPÓSITO EN EL REPOSITORIO INSTITUCIONAL E-PRINTS COMPLUTENSE

Los abajo firmantes, alumno/s y tutor/es del Trabajo Fin de Grado (TFG) en el Grado en ……Ingeniería del Software……de la Facultad de ……Informática……, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TF) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Periodo de embargo (opcional):

- ☐ 6 meses
- ☐ 12meses

TÍTULO del TFG: ……Generación de historias centradas en los personajes mediante simulaciones……

Curso académico: 2015 / 2016

Nombre del Alumno/s:

……Juan Orellana Quemada, Samuel Rodríguez Oliva……
……y Oscar David Vásquez Peraza……

Tutor/es del TFG y departamento al que pertenece:

……Gonzalo Méndez Pozo y Raquel Hervás Ballesteros……
……Departamento de Ingeniería del Software e Inteligencia Artificial……
……

Firma del alumno/s Firma del tutor/es

Agradecimientos

- Porque es lo que hacen los amigos,
siempre se perdonan.
- Ah claro si, tienes toda la razón. Yo te
perdono ... por tu puñalada trapera

Conversación entre Asno y Shrek, Shrek

Completar esta investigación ha sido una arduo trabajo que no podría haberse realizado sin la ayuda de varias personas que nos han apoyado durante el desarrollo de la misma. Por esta razón y a todos vosotros queremos agradeceros vuestra paciencia y atención. Gracias.

Gracias a nuestro director del proyecto, Gonzalo Méndez Pozo, por aguantarnos durante tantas reuniones en las que no sacábamos nada en claro y guiarnos por un camino que de verdad hemos podido recorrer, así como a nuestra co-directora, Raquel Hervás Ballesteros, que siempre nos recibía de buen humor y nos daba ánimos y buenas ideas.

Gracias a nuestras ayudantes, Sara e Irma, por ayudarnos con buenas ideas en el diseño del sistema y un bonito mapa.

Y por último, gracias a nuestros familiares y amigos, por apoyarnos en los momentos difíciles y soportar nuestros agobios, e intentar demostrarnos que si nos esforzábamos llegaríamos a la meta propuesta. Curioso, tenían razón.

Resumen

Entra y cierra la puerta, hace frío ahí fuera

Tabernero del Hearthstone: Heroes of Warcraft

La Generación automática de historias es un camino que empezó a recorrerse a principios de los años 70. El objetivo de este campo es el de dotar de una cualidad humana, la creatividad, a una inteligencia artificial con el fin de que sea capaz de reproducir esta capacidad y demostrar que es competente para narrar una historia. Si bien es cierto que es un campo sobre el que ya se ha trabajado y debatido en innumerables ocasiones, es igual de cierto que las soluciones dadas no cubren el problema en su totalidad. Intentando aportar un granito de arena a esta investigación, este proyecto trata de crear historias a través de personajes independientes e imprevisibles capaces de controlar de manera autónoma y personal sus propias decisiones con el fin de crear historias variadas dentro de un mismo entorno.

Para cumplir este propósito existe un campo concreto que es la Generación de historias a través de Agentes Inteligentes. Estos Agentes representan entidades software con una inteligencia artificial avanzada que les permite percibir su entorno e interactuar con él, comunicándose con otros Agentes mientras tratan de cumplir objetivos específicos que se les presentarán a lo largo de la historia, los cuales provocarán que surjan conflictos de interés entre los distintos Agentes que se resolverán mediante simulaciones, donde se deciden los resultados de estos conflictos que al narrarse generan la historia.

Para poder producir esta idea, se desarrolla una aplicación en Java que mediante un sistema de Agentes Inteligentes consigue a través de la planificación de distintos objetivos crear por medio de simulaciones un número relevante de historias variadas. Para ello, se necesitará de la plataforma JADE, que permite interactuar con los agentes, un planificador externo y un simulador capaz de realizar las susodichas simulaciones.

Siguiendo este proceso, ya explorado en anteriores investigaciones de esta facultad, se propone un sistema de generación de historias variadas capaz de trabajar con un número mayor de Agentes, así como de generar personajes

más profundos y un entorno con mayor complejidad. A través de este sistema se pretende que las historias se produzcan en mundos variados, con numerosos personajes capaces de actuar de manera distinta en cada simulación. Estos personajes contarán con sus propias normas así como sus rasgos y características que los definirán como únicos, serán capaces de interactuar con objetos y tendrán un sistema más variado de generación de frases para reflejarse en la narración, favoreciendo enormemente la diversidad a la hora de generar historias.

A continuación, se exponen las pruebas realizadas y las soluciones obtenidas en el desarrollo de la aplicación y se explica el plan de trabajo, documentando las bases técnicas de las tecnologías usadas así como detallando en profundidad el trabajo realizado, con el objetivo de poder continuar este trabajo de investigación o aportar experiencia para futuras investigaciones relacionadas con el tema.

Palabras Claves

Generación de historias, Inteligencia Artificial, Sistema multiagente, Simulación, Planificación.

Abstract

Automated Storytelling is a long path that began in the early 70's. The objective of this field is to provide a human quality, creativity, to an artificial intelligence system. The final purpose is to prove that the system is able to reach this capability, demonstrating that it is competent to tell a story. Though is true that this field has already been worked on and debated countless times, it is equally true that the solutions given do not cover the whole problem. With the intention of helping the research, this project tries to create independent and unpredictable characters, ir order for them to be able of controlling autonomously and individually their own decisions and therefore create diverse stories within the same environment.

To fulfil this purpose, a specific field called Storyteller System through Intelligent Agents exists. These agents are software entities with an advanced artificial intelligence that allows them to perceive their environment and interact with it, they are able to communicate with other agents as they try to success on the specific objectives that they will have throughout the story. These objectives will result on conflicts of interests between the different agents that will be solved through simulations, so when these simulations get finished they generate the story.

To produced this idea, a Java application is developed through a system of Intelligent Agents and planning different objectives. This application success on create a significant number of diverse stories through simulations. In order to be capable of doing that, is needed the platform JADE, which allows to interact with the agents, an external planner and a simulator, capable of simulate what is needed.

Following this process, which has been explored on previous investigations made in this university, is proposed a storyteller system capable of work with a larger number of agents, as well as generate characters with more possibilities and a more complex environment.

The intention with this system is that the stories take place in different worlds, with many characters capable of acting in different ways in each

simulation. These characters will have their own rules and features that will defined them as unique. They will also be able of interacting with objects and will have a more diverse system to tell their own phrases, greatly favouring diversity in the storytelling.

In the following document, we will discuss the testing and solutions obtained in the development of the application and will proceed to explain the work plan. Documenting the technical basis of the technologies used and detailing in depth the work done is essential, with the objective of continuing this research and contribute with this experience for future researches related to this matter.

Keywords

Storytelling, Artificial Intelligence, Multiagent System, Simulation, Planing.

Índice

Índice de figuras

Índice de Tablas

Lista de Códigos

Capítulo 1

Introducción

Caminante, no hay camino, se hace
camino al andar.

Antonio Machado

La generación de historias de manera artificial plantea algunos problemas que se resumen, sobre todo, en saber hasta qué punto es capaz de imitar la inteligencia artificial a la inteligencia humana. Más concretamente, la característica humana que más destaca a la hora de crear una historia es la creatividad. La creatividad, como tal, es un concepto abstracto difícil de explicar. Según la RAE es "la capacidad de crear", y para crear se debe ser original, lo que obliga a que la creatividad deba innovarse y reciclarse cada pocos años. Es, por tanto, una característica difícil de encontrar y reproducir entre las propias personas, pues no todas tienen la capacidad de ser creativas y, mucho menos, de narrar una historia interesante.

¿Es pues la creatividad una característica que podemos aportar a la inteligencia artificial? Sí y no. La inteligencia artificial no es capaz de crear por sí sola, pues siempre está parametrizada y limitada por las características que le proporcionó su programador. La creatividad no se puede conseguir en la inteligencia artificial, pero si que se puede simular. A través de simular la creatividad, se pretende crear historias variadas con un mismo *software*. Y para crear las diferentes historias tenemos numerosas opciones.

La primera manera de simular la creatividad es haciendo que se creen historias completamente distintas, proponiendo, por ejemplo, lugares distintos donde se ejecuta la acción, como puede ser un mundo medieval fantástico o uno con características tipo *Western*. También se puede conseguir a través de personajes totalmente distintos. Siguiendo el ejemplo, en una historia habría caballeros y dragones y en la otra *sheriffs* y bandidos. Gracias a esta manera el numero posible de historias a generar aumenta considerablemente.

Sin embargo no es la única manera de conseguir esto, también se busca que dos historias con las mismas premisas resulten distintas. Si todas las

historias tienen las mismas características, la única manera posible de hacer que produzcan resultados diferentes es buscar un momento en el cual divergen. Este momento consideramos que debe producirse cuando los personajes entran en conflicto en la historia, así, según las características de cada personaje o si simplemente aparecieron en el momento más oportuno, la historia tomará un curso u otro.

Ante estas posibilidades, se proponen dos soluciones. La primera, es proponer una aplicación altamente configurable donde el usuario pueda elegir los detalles de la historia, como los nombres de los personajes o el mundo planteado. De la misma manera, se propone la posibilidad de contar con un gran abanico de personajes posibles, dividiendo a estos personajes en clases distintivas, de manera que distintos personajes se comporten de distintas maneras y produzcan historias nuevas.

La segunda, es conseguir que cada elemento de la aplicación sea único. Para ello, se dota a los personajes de características propias que los hacen diferentes entre sus iguales, así como de la posibilidad de conseguir objetos que les puedan ayudar en los distintos conflictos en los que se vean envueltos. Gracias a estos cambios, se introducen en los conflictos entre personajes, poniendo una batalla como el ejemplo mas típico, tiradas de dados al mas puro estilo rolero, que se ven modificadas por estas cualidades. De esta manera, ciertamente algunos personajes tienen más posibilidades de vencer que otros y el factor "suerte" siempre dota de una particularidad especial a cada historia.

Para poder reproducir esta idea, se necesita de *Agentes Inteligentes*, unidades software con una inteligencia artificial avanzada a los cuales se les puede proponer un objetivo que siempre intentarán cumplir, que harán las veces de personajes. Para que estos Agentes puedan ser capaces de entender su objetivo e interpretarlo con respecto al mundo en el que simulan vivir, se requiere de un planificador, una herramienta capaz de proponerle a cada Agente en específico los pasos que debe seguir en pos de cumplir sus objetivos. En numerosas ocasiones esto lleva a que los Agentes entren en conflicto entre ellos, pues a veces los objetivos de un Agente son opuestos a los de otro. Para paliar este problema, se cuenta con la ayuda de un simulador, una herramienta capaz de simular, por ejemplo, una batalla entre distintos personajes para poder ver así cual saldrá vencedor y, por tanto, seguir con su plan para cumplir su objetivo.

Enfocándolo de una manera más simple, imaginemos que la historia es como el rodaje de una película. Los agentes conforman los personajes o actores de esta, cada uno con sus propios intereses y motivaciones. El planificador será el equipo de guionistas, decidiendo que camino deberá tomar cada personaje para cumplir su cometido en la historia. Y por último, la simulación, hará las veces de director,decidiendo en las escenas importantes que personaje seguirá adelante en la historia y qué personaje será descartado (por

ejemplo, en una batalla en cada simulación el bando victorioso puede variar) centrando a partir de ahí la historia otra vez en los personajes que siguen disponibles y pidiendo al equipo de guionistas, el planificador, que determinen cuál será el siguiente camino a tomar, hasta que nuevamente se tenga que volver a simular o dirigir.

Así pues, se explican a continuación las características del proyecto, todo el material necesario para comprender su funcionamiento y las cualidades necesarias que debimos proporcionar con el ánimo de que este fuese capaz de cumplir nuestras expectativas. Para ello este documento se estructura de la siguiente manera:

1.1. Estructura General del Documento

- **Capítulo 1: Introducción.**

  Capítulo en el que se introduce de manera general la investigación desarrollada.

- **Capítulo 2: Estado de la Cuestión.**

  Capítulo en el que se explica en profundidad las tecnologías usadas en la investigación, así como el funcionamiento de las mismas

- **Capítulo 3: Objetivos.**

  Capítulo que puntualiza de manera concreta los objetivos marcados para esta investigación.

- **Capítulo 4: Primeros Pasos.**

  Capítulo que relata la experiencia en las primeras fases de la investigación y los problemas encontrados, de manera que se pueda entender el desarrollo de la misma.

- **Capítulo 5: Arquitectura del Generador de historias.**

  Capítulo que ofrece un conocimiento avanzado sobre la arquitectura del sistema.

- **Capítulo 6: Mapa.**

  Capítulo en el que se describe como se genera el mundo donde trascurre la historia, personificado en un mapa, así como su diseño e implementación.

- **Capítulo 7: Personajes.**

  Capítulo en el que se ve en profundidad el diseño y la implementación de los distintos personajes, así como de todas las cualidades que los definen.

- **Capítulo 8: Objetos.**

  Capítulo en el que se detalla la creación e implementación de los objetos que usarán los personajes, así como la manera de encontrarlos y usarlos.

- **Capítulo 9: Simulaciones.**

  Capítulo en el que se exponen diferentes ejemplos del funcionamiento del software creado.

- **Capítulo 10: Trabajo Individual.**

  Capítulo que detalla el trabajo individual realizado por cada miembro del equipo en la investigación.

- **Capítulo 11: Conclusiones.**

  Capítulo en el que se exponen las diferentes conclusiones en las que ha derivado la investigación.

- **Capítulo 12: Trabajo Futuro.**

  Capítulo que propone ciertas ideas para proseguir trabajando sobre la misma investigación.

Introduction

Computational Storytelling raises some problems, which are summarized in how precisely can imitate an artificial intelligence system the human intelligence. More specifically, the human characteristic that stands out when creating a story is creativity. Creativity is an abstract concept difficult to explain. According to RAE it is "the ability to create", which means that needs to be original. These requires for creativity to innovate and recycle itself every few years. It is, therefore, a difficult feature to find among people, since not everyone has the ability to be creative and, much less, to tell an interesting story.

Is, therefore, creativity a feature that can be given to an artificial intelligence system? Yes and no. An artificial intelligence system is not capable by itself to create, because it is always settled and limited by the features provided by his programmer. Creativity can not be achieved on the artificial intelligence, but it can be simulated. Through simulating creativity, we are able to create different stories with the same software. And to do so we have lot of options.

The first way to simulate creativity is by making completely different stories, like, for example, proposing different places in which the story takes place, such as a fantastic medieval environment or a Western scenario. It can also presents totally different characters. Following the previous example, in one story you will got knights and dragons and in the other one sheriffs and bandits. Thanks to this, the number of stories you can get increases.

However, this is not the only way to achieve this, you can also seek that two stories with the same assumptions turn out to be different. If two stories have the same characteristics, the only way to make them develop into two different stories is to find a moment in which them diverge. We currently believe that this moment should happen when the characters conflict in the story. This way, considering the features of each character or just if they appear on the right time, the story will take one way or another.

Given these possibilities, two solutions are proposed. The first one is to expose a highly configurable application where the user can choose the details of the story, as well as the names of the characters or the details of the world designed. Similarly, the possibility of having a wide range of

possible characters is also an option, dividing these characters in several classes, so that different characters will behave in different ways and produce new stories.

The second way is to make every element of the application unique. To do this, characteristics are given to the characters in order to make them different from their peers, as well as the possibility of getting objects that may help them in the different conflicts that will involved them. These changes introduce conflicts between the characters, with a battle as the most typical example, making them dice rolls like in role games. These dice rolls are modified by these qualities. In this way, some characters are more likely to win than others and the factor of "luck" always gives a special feature to every story.

In order to get this idea, the use of Intelligent Agents is needed. These software units with an advanced artificial intelligence can be proposed with a goal that they will always try to reach. These Intelligent Agents will play the role of the different characters. For these Agents to understand their targets and interpret them rightly in the world they simulate to live in, a planner is required This planner is a tool able to propose to each Agent the specific steps that they should follow in order to fulfil their objectives. This leads to the agents having conflicts between them, because sometimes the goals of an Agent are opposite to the goals of one another. To solve this problem, a simulator is needed, a tool able to simulate a battle, for example, between different characters to see which shall become the winner so he can keep on with his plan to succeed on its objective.

In an easier way, imagine that the story is like rolling a film. The agents represent the characters or actors of the film, each with its own interests and motivations. The planner will be the writing team, deciding which way each character should take in order to fulfil its role in the story. And finally, the simulation will act as the manager, deciding on important scenes which character will go forward in the story and which will be excluded (like in a battle, in each simulation the winner may be different). These way the story focus again on the characters that remained available. The simulation will ask the writing team, the planner, to determine what the next course will be, until once again it is needed to re-simulate.

Further on, the characteristics of the project are explained, as well as all the necessary tools that we need to understand its functioning and the qualities we should provide hoping that it would be able to meet our expectations. To do so, this memory is structured as follows:

General Structure of the Document

- **Capítulo 1: Introducción.**

  In this chapter the investigation is introduced.

- **Capítulo 2: Estado de la Cuestión.**

  Chapter in which the technologies used on the research are explained, as well as how they work.

- **Capítulo 3: Objetivos.**

  Chapter that indicates specifically the objectives set for this research.

- **Capítulo 4: Primeros Pasos.**

  Chapter that describes the experience in the early stages of the research and the problems founded, making possible to understand the development of the investigation.

- **Capítulo 5: Arquitectura del Generador de historias.**

  Chapter that provides an advanced knowledge of the system architecture.

- **Capítulo 6: Mapa.**

  Chapters that describes the world where the story takes place, write out as a Map, as well as its design and implementation.

- **Capítulo 7: Personajes.**

  Chapter where the design and implementation of the different characters in the story is explained, as well all the qualities that define them.

- **Capítulo 8: Objetos.**

  Chapter in which the objects development and how different characters can use them is detailed.

- **Capítulo 9: Simulaciones.**

  Chapter in which different output examples of the created software are exposed.

- **Capítulo 10: Trabajo Individual.**

  Chapter in which is detailed the work performed by each team member on the research.

- **Capítulo 11: Conclusiones.**

  Chapter in which the different conclusions which has led the investigation are set.

- **Capítulo 12: Trabajo Futuro.**

  Chapter in which some ideas to continue working on the same research get proposed.

Capítulo 2

Estado de la Cuestión

El trabajo del maestro no consiste tanto en enseñar todo lo aprendible, como en producir en el alumno amor y estima por el conocimiento

John Locke

La finalidad de éste capítulo es ofrecer una base de conocimientos sólida donde queden reflejados los distintos conceptos y herramientas sobre los que se apoya el trabajo realizado. Para ello se ha estructurado este capitulo en tres partes fundamentales. Diferenciar cada parte permite elaborar una imágen clara sobre los pilares que constituyen el trabajo realizado.

La primera parte cuenta un resumen de los conceptos y las tecnologías referentes al sistema de Agentes. Los Agentes son el núcleo principal sobre el cuál se basa este proyecto, siendo estos los "actores" de cada historia.

En la segunda parte se describe los sistemas planificadores y el lenguaje que se ha usado para que los Agentes software sean capaces de realizar una planificación. La planificación es el sistema que van a usar estos Agentes para ser dotados de cierta "inteligencia" e interactúen con todo su entorno.

La última parte está dedicada a la narrativa computacional, en la cual se verán otros sistemas de generación de historias relacionados con el ámbito del proyecto. Entender y conocer el ámbito de la narrativa computacional, la creatividad computacional y la generación de historias, permite tener mejor enfoque sobre el contenido del proyecto y el punto desde el que parte.

2.1. Agentes

2.1.1. Agente Software

Wooldridge (2009) Define el concepto de Agente de la siguiente manera:

> *Un agente es un sistema informático que se encuentra en un entorno, y que es capaz de realizar acciones autónomas en éste entorno con el fin de cumplir sus objetivos.*

Con ésto se quiere hacer ver que un Agente en particular es una entidad propia, que poseerá características y ciertas condiciones que le definirán como un individuo único dentro de su entorno. Un Agente software dispone de todas estas características, y además es capaz de realizar acciones por cuenta propia en función de las situaciones en las que se encuentre en un momento dado.

2.1.2. Agentes Inteligentes

Considerar que un Agente sea inteligente sugiere preguntarse, ¿cuándo considerar a un Agente inteligente? Wooldridge (2009) propone una lista de características que puede poseer un Agente Inteligente

- Reactividad: Agentes Inteligentes capaces de percibir su entorno y responder a los cambios que ocurren en éste con el fin de satisfacer sus objetivos.
- Proactividad: Agentes Inteligentes capaces de manifestar comportamientos orientado a objetivos tomando la iniciativa con el fin de satisfacer sus objetivos .
- Habilidades sociales: Agentes Inteligentes capaces de interactuar con otros Agentes(y posiblemente humanos).

Realizar una generación de historias rica en contenido, calidad y diversidad requiere usar una combinación de estos agentes, ya sean solo agentes reactivos, proactivos, ó en algún caso una mezcla de ambos con el fin de satisfacer sus objetivos.

2.1.3. Sistemas Multi-agente

Según Bellifemine et al. (2007) :

> *Incluso si un sistema de agentes puede basarse en un único agente funcionando en un entorno e interactuando con usuarios si fuese necesario, normalmente estos sistemas consisten en multiples agentes. Estos Sistemas Multi-Agente (SMA) pueden modelar sistemas complejos e introducir la posibilidad de que los agentes tengan objetivos en común ó en conflicto. Estos agentes pueden interactuar entre ellos indirectamente (alterando el entorno) ó directamente (vía comunicación y negociación). Los agentes pueden decidir si cooperan para un beneficio mutuo ó si compiten por sus propios intereses.*

Una historia en la que sólo actúa un personaje a priori no disfruta de tantos elementos, eventos y acontecimientos como lo puede hacer una en la que coexistan varios personajes. Por tanto, un sistema multi-agente se puede ver como un entorno en el que coexisten varios agentes y donde estos serán capaces de interactuar entre sí.

2.1.4. Foundation for Intelligent Physical Agents - FIPA

FIPA (2016) es una organización que promueve las tecnologías basadas en agentes y la interoperatividad de sus estándares con otras tecnologías.

En el libro Mas (2005) mencionan sobre FIPA lo siguiente:

> *El estándar FIPA define los servicios que debe proporcionar toda plataforma de agentes: un sistema encargado del transporte de mensajes (Internal Platform Message Transport), un sistema de gestión de agentes (Agent Management System), un servicio de páginas amarillas (Directory Facilator) y un canal de comunicaciones para los agentes (Agent Communication Channel). Cada uno de estos servicios , excepto el de transporte de mensajes, es suministrado por agentes especializados, lo cual supone que la comunicación con ellos será mediante mensajes ACL (Agent Communication Language) mediante la ontología definida para ese servicio.*

En FIPA (1997) se dispone de información detallada sobre los mensajes FIPA ACL.

La Tabla 2.1 nos muestra la lista de parámetros que caracterizan a los mensajes FIPA ACL.

Esta tabla y una explicación más detallada de todos los parámetros son consultables en FIPA (2002).

2.1.5. Mensajes FIPA

En la tabla 2.1, se ve cuáles son los parámetros que caracterizan a los mensajes FIPA ACL. Cuando los agentes se comunican, identificar el tipo de mensajes que se envían es muy importante de cara al control y tipo de tareas que quieren realizar los Agentes. La Tabla 2.2 contiene algunos de los diferentes tipos de comunicación que ofrece el campo `performative`:

Parámetros	Categoría del parámetro
`performative`	Tipo de acto comunicativo
`sender`	Participante en la conversación
`receiver`	Participante en la conversación
`reply-to`	Participante en la conversación
`content`	Contenido del mensaje
`language`	Descripción del contenido
`enconding`	Descripción del contenido
`ontology`	Descripción del contenido
`protocol`	Control de conversación
`conversation-id`	Control de conversación
`reply-with`	Control de conversación
`in-reply-to`	Control de conversación
`reply-by`	Control de conversación

Tabla 2.1: Tabla parámetros de los mensajes FIPA ACL

Acto comunicativo	Descripción
`Accept Proposal`	La acción de aceptar una propuesta
`Call for Proposal`	La acción de hacer propuestas para realizar cierta acción
`Confirm`	El emisor informa al receptor que una proposición dada es cierta
`Disconfirm`	El emisor informa al receptor que una proposición dada es falsa
`Failure`	La acción de indicar a otro agente que se intentó realizar una acción pero este intento falló
`Inform`	El emisor informa al receptor que una proposición dada es cierta
`Propose`	La acción de enviar una propuesta para realizar cierta acción, dadas ciertas precondiciones
`Refuse`	La acción de rehusarse a realizar una acción dada, y explicando la razón por la cual no se realiza la acción
`Reject Proposal`	La acción de rechazar la propuesta de realizar cierta acción durante la negociación
`Request`	El emisor solicita al receptor a realizar cierta acción. Una importancia de `Request` es que solicita al receptor realizar otro acto comunicativo

Tabla 2.2: Tipos de actos comunicativos definidos en `performative`

En el cuadro de Código 2.1[1] se puede ver un ejemplo del contenido de un mensaje usando el acto `Request`:

Código 2.1: Ejemplo de mensaje FIPA ACL

```
(request
:sender (agent-identifier :name alice@mydomain.com)
:receiver (agent-identifier :name bob@yourdomain.com)
:ontology travel-assistant
:language FIPA-SL
:protocol fipa-request
:content
""((action
(agent-identifier :name bob@yourdomain.com)
(book-hotel :arrival 15/10/2006
:departure 05/07/2002 ... )
))""
)
```

2.1.6. Modelo del gestor de agentes

El gestor de agentes proporciona el framework donde los agentes FIPA existen y operan. Éste establece el modelo lógico de referencia para la creación, registro, localización, comunicación, migración y retiro de los agentes.

Las entidades contenidas en la Figura 2.1 son una configuración lógica y no implica ninguna configuración física.

[1] Recuperado de Bellifemine et al. (2007)

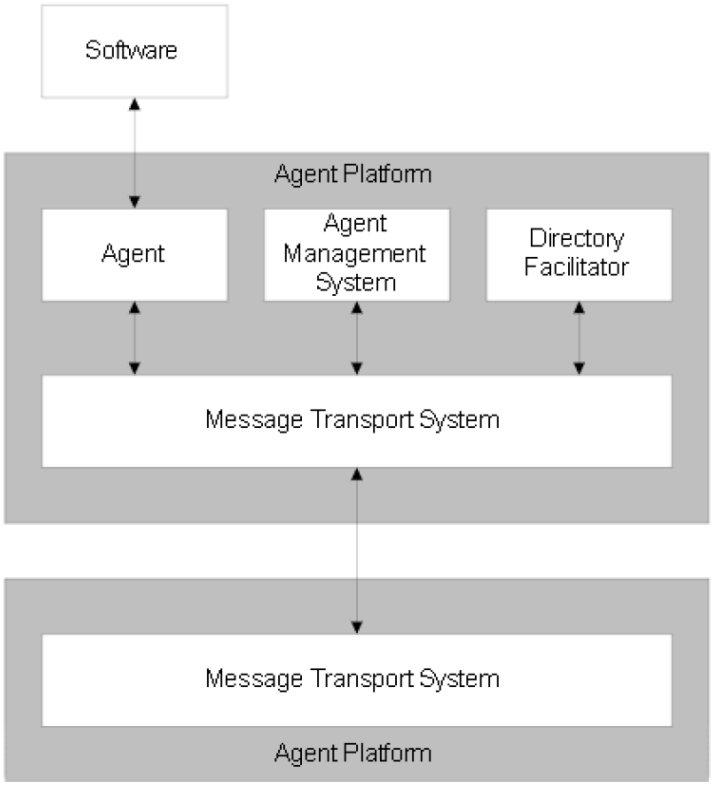


Figura 2.1: Modelo de referencia del gestor de agentes.

Recuperado de http://www.fipa.org/specs/fipa00023/XC00023H.html.

Una **Agent Platform (AP)** proporciona la infraestructura física donde los agentes son desplegados. La AP comprende la máquina/s, sistema operativo, software de soporte de agentes, componentes de gestión de agentes FIPA (DF, AMS y MTS) y los agentes.

El **Directory Facilitator (DF)** es un componente esencial del AP(explicado más adelante). El DF proporciona un servicio de *páginas amarillas* para el resto de agentes. Los agentes pueden registar sus servicios en el DF ó consultar éste para conocer que servicios ofrecen otros agentes. La Figura 2.2 muestra un ejemplo de este servicio.

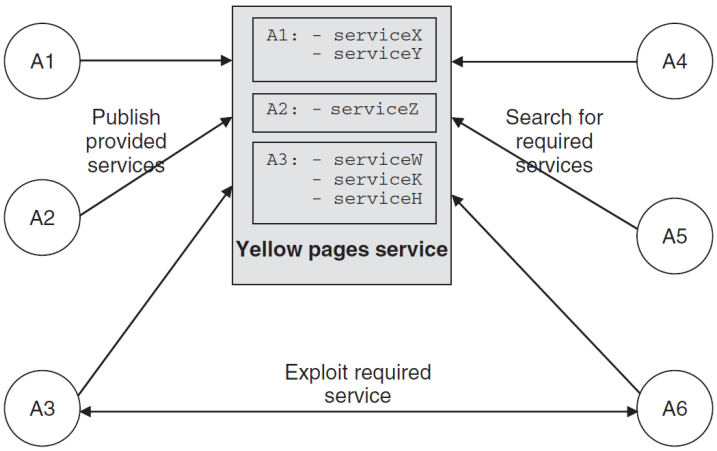


Figura 2.2: Servicio de páginas amarillas.

Recuperado de Bellifemine et al. (2007).

El **Agent Management System (AMS)** es un componente esencial del AP. El AMS ejerce control de supervisión sobre el acceso y uso del AP. Sólo un AMS debe existir para cada AP. El AMS mantiene un directorio de AIDs (Agent Identifications) que contienen direcciones de transporte (entre otras cosas) para los agentes registrados en el AP. El AMS ofrece un servicio de *páginas blancas* para otros agentes. Cada agente debe registrarse en el AMS para poder obtener un AID válido.

Un **Message Transport Service (MTS)** es el método de comunicación por defecto entre agentes a través de diferentes APs.

El diseño interno de una AP es labor de los desarrolladores del sistema de agentes y no está sujeto a una estandarización por parte de FIPA. Las APs y los agentes que son nativos a esa AP, ya sea por haber sidos creados ó migrados a esa AP, pueden utilizar cualquier método propio para la comunicación interna entre ellos.

2.1.7. Identificación de los agentes

Es oportuno comentar que para que un agente disponga de AID válido, el nombre usado como identificador al momento de crear un agente ha de ser único dentro del AP. De otra manera el sistema considera que el identificador se encuentra repetido y no será posible registrar correctamente el agente dentro del AP. El nombre identificador no es sensible a mayúsculas ni minúsculas, y es altamente recomendable que no contenga espacios en este para asegurar el correcto funcionamiento del sistema.

2.1.8. Ciclo de vida de los agentes

Un agente FIPA existe físicamente dentro de una AP y usa las facilidades proporcionadas por ésta para ejecutar sus funcionalidades. Dicho esto, el agente posee un ciclo de vida que es controlado por la AP como podemos observar en la Figura 2.3.

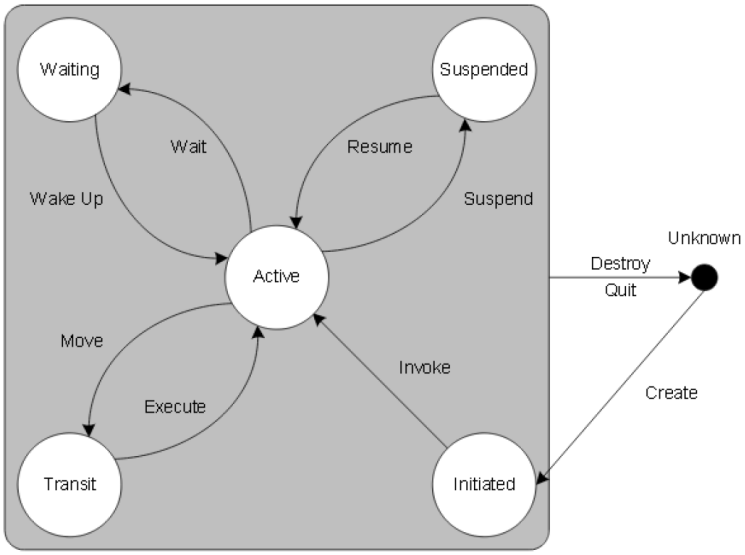


Figura 2.3: Ciclo de vida de los agentes FIPA.

Recuperado de http://www.fipa.org/specs/fipa00023/XC00023H.html.

Active: El MTS entrega mensajes al agente con normalidad.

Initiated/Waiting/Suspended: El MTS bien almacena en un buffer los mensajes hasta que el agente vuelva al estado activo ó reenvía los mensajes a una nueva localización (si el reenvío está disponible para el agente)

Transit: El MTS bien almacena en un buffer los mensajes hasta que el agente vuelva a estar activo (eso es, la función mover a fallado en la AP original ó el agente a iniciado satisfactoriamente en la AP destino) ó reenvía los mensajes a una nueva localización (si el reenvío está disponible para el agente). Téngase en cuenta que sólo los agentes móviles pueden entrar en estado de tránsito. Esto asegura que un agente estático ejecuta todas sus instrucciones en el nodo donde fue invocado.

Unknown: El MTS bien almacena en un buffer los mensajes o los rechaza, dependiendo de la política del MTS y de los requerimientos de transporte del mensaje.

Un agente en JADE dispone de 2 métodos fundamentales:

- `setup()`: Método que se invoca al momento que se crea el agente, cuando el agente pasa a los estados de *iniciado* y *activo*. Dentro de

este método se especifican las instrucciones y los comportamientos de los que dispondrá el agente.

- `takeDown()`: Método invocado automáticamente al momento que se destruye el agente. Principalmente nos permite liberar recursos usados por el agente antes de que deje de existir en la AP.

2.1.9. Java Agent Development Framework - JADE

Bellifemine et al. (2008), JADE es el middleware desarrollado por TI-LAB[2] para el desarrollo de aplicaciones multi-agente distribuidas basadas en comunicación peer-to-peer. La inteligencia, la iniciativa, la información, los recursos y el control pueden ser totalmente distribuidos en terminales móviles así como en ordenadores en entornos cerrados. JADE está completamente desarrollado en Java y es la plataforma software que nos suministra las herramientas necesarias para poder desarrollar una aplicación basada en el paradigma de agentes.

La Figura 2.4 muestra los elementos principales de la arquitectura de una plataforma JADE Bellifemine et al. (2007).

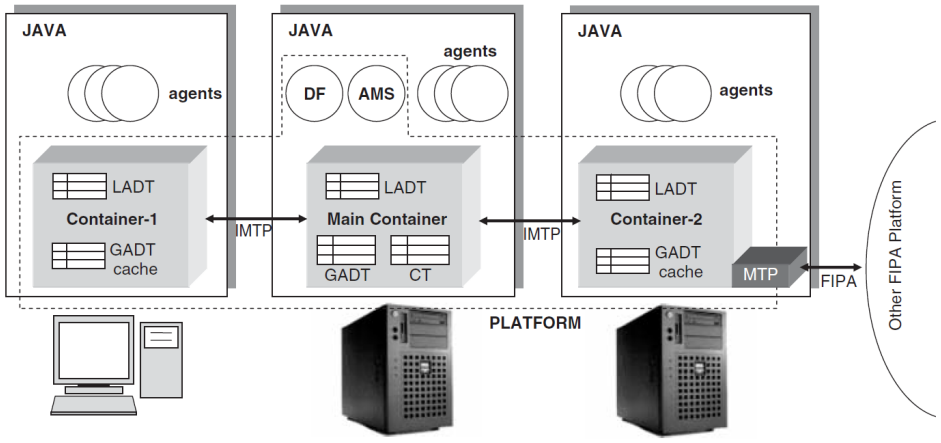


Figura 2.4: Relación entre los principales elementos de la arquitectura

2.1.10. Comportamientos JADE

Los agentes en JADE se especifican mediante comportamientos. Los comportamientos definen la forma de actuar para cada agente ante un problema dado. Cada comportamiento se encarga de resolver una tarea, comunicarse, o evaluar alguna condición del agente.

[2] `http://jade.tilab.com/`

Cada agente dispone a su vez de una cola de comportamientos en la cual ser irán incluyendo los comportamientos agregados con `addBehaviour()`. Cuando llega el turno de ejecución de un comportamiento, este intenta realizar su tarea y al finalizar es removido y eliminado de la cola. Existen comportamientos que en función del tipo que sean, al finalizar su tarea intentarán añadirse nuevamente a la cola.

Los tipos de comportamientos en JADE son los siguientes:

- **Behaviour**. Comportamiento básico que proporciona los métodos `action()` [3] y `done()`[4].
  - **SimpleBehaviour**. Comportamientos que realizan tarea simples.
    - **OneShotBehaviour**. Tipo de comportamiento que su método `action()` se ejecuta una única vez.
    - **TickerBehaviour**. Comportamiento de ejecución periódica.
    - **WakerBehaviour**. Es una variación del OneShotBehaviour que inicia su ejecución después de un tiempo especificado.
    - **CyclicBehaviour**. Comportamiento cuyo método `done()` siempre devuelve `false`, por tanto al terminar su `action()` vuelve a la cola de comportamientos.
  - **CompositeBehaviour**. Nos permite integrar dentro de su estructura múltiples comportamientos.
    - **ParallelBehaviour**. Este tipo permite la ejecución en paralelo de varios comportamientos.
    - **SerialBehaviour**. Nos permite representar sucesiones de comportamientos.
      - **SequentialBehaviour**. Obliga a los comportamientos que lo integran a ejecutarse en un orden especificado.
      - **FSMBehaviour**. Nos permite emular una máquina de estados finita basada en comportamientos.

La Figura 2.5 nos muestra un diagrama detallado de los comportamientos en JADE.

[3] Dentro se especifican las tareas que va a realizar el comportamiento.
[4] Devuelve `true` si el comportamiento ha finalizado su ejecución.

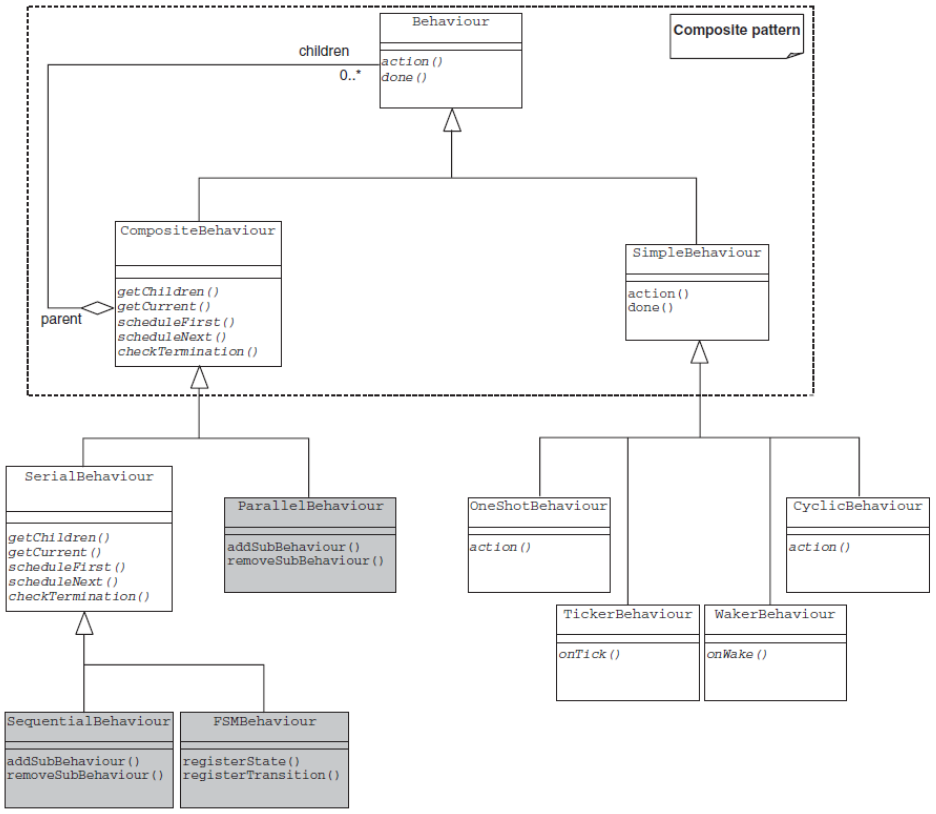


Figura 2.5: Diagrama de jerarquía de los comportamientos en JADE.

Recuperado de Bellifemine et al. (2007).

Al momento de diseñar el comportamiento de un agente es probable que se requiera de comportamientos que sean capaces de ejecutarse en paralelo y al mismo tiempo. El tipo `ParallelBehaviour` hace una ejecución en paralelo de dos ó más comportamientos pero lo que hace realmente es ejecutar un método `action()` a la vez, del comportamiento que toque, y luego va alternando con los demás comportamientos.

Esto se debe a que el manejo de los comportamientos no sigue una política de expulsión, es decir, cuando se está ejecutando el método `action()` de un comportamiento éste nunca se ve interrumpido para que otro inicie su ejecución. Sólo cuando ése método termina y devuelve la salida `done()` como `true` se otorga el control al `action()` del siguiente comportamiento en la cola.

JADE ofrece una solución bastante sencilla si necesitamos que dos comportamientos tengan la oportunidad de realmente ejecutarse a la vez, esto es asignando comportamientos a hilos dedicados.

La clase `ThreadedBehaviourFactory` dispone de un método `wrap()` que se encarga de encapsular el comportamiento en un hilo dedicado cuando se

añade éste a un agente.

Por último, en la Figura 2.6 tenemos un diagrama de flujo que representa el ciclo de ejecución de un agente con sus comportamientos en JADE.

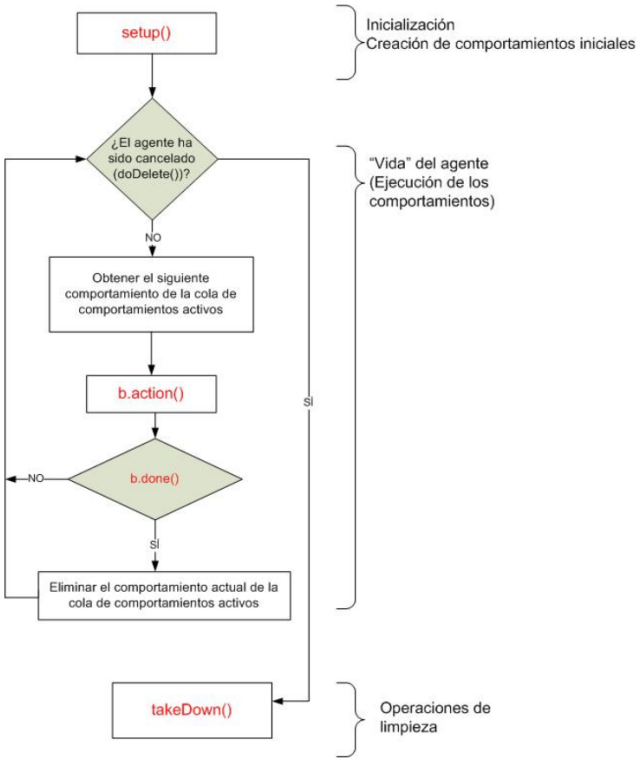


Figura 2.6: Flujo de ejecución de un agente y sus comportamientos en JADE.

Recuperado de Bellifemine et al. (2007).

2.2. Planificación y Lenguaje

2.2.1. Planificación

Desde la perspectiva de Hoffmann (2011) :

> *La planificación es el problema de seleccionar una serie de acciones orientadas a objetivos, basados en una descripción de alto nivel del mundo. Desde la perspectiva de la informática, la planificación es sólo un formalismo que describe brevemente máquinas de estados grandes, similares a redes de autómatas o máquinas de Turing.*

Mediante el proceso de planificación se consigue que los actores de las historias generadas logren disponer de una serie de acciones, que realizaran secuencialmente, y así se aseguran que cada personaje puede cumplir su objetivo.

2.2.2. JavaFF

En el artículo Hoffmann y Nebel (2001) *The FF Planning System: Fast Plan Generation Through Heuristic Search*, se describen y evalúan las técnicas algorítmicas que son usadas en los sistemas de planificación FF [5]

En Coles et al. (2008) se explica que JavaFF es una implementación en Java de la estructura básica de Hoffmann y Nebel (2001), basada en en código fuente de CRIKEY Coles et al. (2009)[6].

El planificador usado para generar las historias está basado en la implementación antes mencionada y, más específicamente, en una leve modificación realizada en Laclaustra et al. (2014) sobre esa implementación.

2.2.3. Planning Domain Definition Language - PDDL

PDDL es un lenguaje para la especificación de problemas creado para la competición de planificadores AIPS-98. Aeronautiques et al. (1998) es un manual que explica la sintaxis y la semántica de este lenguaje. De igual manera cuenta que PDDL está hecho con la intención de expresar la "física" de un dominio, es decir, que predicados existen, que acciones son posibles y cuales son los efectos que ocurren tras cada acción.

Un aspecto importante del funcionamiento de este lenguaje, en conjunto con un planificador que sea capaz de interpretar las instrucciones especificadas, es que necesariamente se debe especificar al menos 2 documentos:

- Dominio: Es el documento que contendrá la especificación de las acciones, los tipos, requerimientos y predicados para poder resolver un problema dado. Es lo que se encarga de dar el conjunto de requerimientos que permiten al planificador procesar un plan.
- Problema: Es el documento que contiene el estado inicial, los objetos y la especificación de objetivos.

[5] Fast Forward. En Hoffmann (2001) nos explican con detalle este sistema.

[6] La referencia que se hace al código fuente es de un documento de Agosto de 2008 que estaba por ser publicado en un recopilatorio en Enero de 2009, el cual es el documento final que referenciamos

2.3. Narrativa Computacional

Dentro del campo de la narrativa computacional hay algunos proyectos que tienen especial interés para el proyecto actual. Estos sirven para evitar partir desde cero, tener conocimiento del estado actual de la cuestión y así tener un punto de partida para aportar algo más a este campo.

2.3.1. NOVEL WRITTER

De los sistemas generadores de narrativa el que presume de ser el primero es el NOVEL WRITER de Klein (1973). Este sistema genera una historia de novela negra en la que se producen asesinatos en una fiesta durante un fin de semana. Para generar las historias utiliza un modelo de microsimulaciones en donde los comportamientos de los personajes y la sucesión de eventos que se producen son llevadas a cabo por normas probabilísticas que cambian constantemente el estado del mundo, mientras que la decisión de quien es el asesino o victima depende de los rasgos de entrada del personaje.

2.3.2. TALE-SPIN

Creado por Meehan (1977), TALE-SPIN es un sistema que narra historias sobre la vida de las criaturas en el bosque. El sistema trabaja sobre planificaciones, por lo que cada personaje tiene un objetivo, en el cual puede participar más de un personaje para su resolución. Además estos objetivos sirven como desencadenadores de acciones de los personajes. El plan que desarrolla el sistema sirve como el hilo conductor de la historia. TALE-SPIN sirvió como punto de partida a proyectos como AUTHOR de Dehn (1981) o a UNIVERSE de Lebowitz (1985).

2.3.3. AUTHOR

Es un programa que genera historias que pretende simular la mente de un autor cuando inventa una historia. Las cuatro fuerzas principales que impulsan el proceso de generación de la historia, de acuerdo con el modelo AUTHOR, son:

- La intencionalidad del autor.
- La reformulación conpetual.
- El recuerdo.
- La oportunidad de mejorar los meta-objetivos.

2.3.4. UNIVERSE

Es el primer sistema generador de historias que centra la narrativa en torno a los personajes. Esta enfocado en la generación de historias como pueden ser las telenovelas o seriales televisivos, en donde la historia gira en las tramas que van surgiendo entre personajes, más que en alcanzar un objetivo final, por lo que se generan historias que están en constante crecimiento.

Un punto interesante en UNIVERSE es la forma de creación tanto de los personajes como de las tramas(Véase en las siguientes Figuras).

```
Name: LIZ CHANDLER (LIZ)
Marriages:
   DON CRAIG [DON] [&MF1] [1980]
   TONY DIMERA [TONY] [&MF3]
IPRs:
   HUSBAND-WIFE     TONY DIMERA [TONY]            0/-8//8/8//6/-6//7/-3
   EX-SPOUSES       DON CRAIG [DON]               -5/-5//4/4//0/0//4/4
Stereotypes: ACTOR KNOCKOUT SOCIALITE PARTY-GOER
Trait modifiers: (SEX F) (AGE YA) (WEALTH 3) (PROMISCUITY -3)
                 (INTELLIGENCE 3)
Overall description:
WEALTH              8
PROMISCUITY         3
COMPETENCE          NIL
NICENESS            0
SELF-CONF           6
GUILE               7
NAIVETE             7
MOODINESS           6
PHYS-ATT            7
INTELLIGENCE        7
GOALS               (FIND-HAPPINESS BECOME-FAMOUS MEET-FAMOUS-PEOPLE)
AGE                 YA
SEX                 F
```

Figura 2.7: Representación de un personaje en UNIVERSE

```
PLOT FRAGMENT: forced-marriage

CHARACTERS:  ?him ?her ?husband ?parent

CONSTRAINTS: (has-husband ?her)          {the husband character}
             (has-parent ?husband)       {the parent character}
             (< (trait-value ?parent 'niceness) -5)
             (female-adult ?her)
             (male-adult ?him)

GOALS: (churn ?him ?her)  {prevent them from being happy}

SUBGOALS: (do-threaten ?parent ?her "forget it") {threaten ?her}
          (dump-lover ?her ?him) {have ?her dump ?him}
          (worry-about ?him)     {have someone worry about ?him}
          (together * ?him)      {get ?him involved with someone else}
          (eliminate ?parent)    {get rid of ?parent (breaking threat)}
          (do-divorce ?him ?her) {end the unhappy marriage}
          (or (churn ?him ?her)     {either keep churning or}
              (together ?her ?him)) {try and get ?her and ?him back together}
```

Figura 2.8: Representación de una trama en UNIVERSE

2.3.5. MINSTREL

Turner (1992) en su tesis doctoral trata el tema de Creatividad Computacional y la generación de narrativa. Por ello realizó este programa, el cual genera historias breves de no más de una página, sobre los Caballeros de la Mesa Redonda del Rey Arturo. Para la generación de la historia primero se realiza una fase de planificación y a continuación una fase de la resolución del problema.

Para la resolución del problema MINSTREL implementa una heurística denominada TRAM (Transform-Recall-Adapt Methods). Cada TRAM integra soluciones a problemas sencillos y al aplicar un conjunto de estas soluciones servirá para solucionar el problema inicial.

2.3.6. MEXICA

MEXICA es un modelo computacional diseñado para estudiar el proceso creativo realizado por Pérez y Pérez (1999). MEXICA genera historias cortas sobre los mexicas, lo primeros habitantes de Méjico. El programa consta dos rutinas. En la primera, el usuario define un conjunto de historias anteriores mientras que en la segunda genera nuevo material, le impone restricciones y revisa que se cumplan estas restricciones para crear nuevas historias. Fue pionera dentro de la generación de narrativa ya que contemplaba, durante el transcurso de la historia, los vínculos emocionales, las tensiones y las posiciones entre los personajes.

2.3.7. BRUTUS

BRUTUS es un programa capaz de generar historias de traición, intriga y algo de misterio en correcta prosa Inglesa.

Sin embargo, gracias a la exposición de llevado a cabo por Bringsjord y Ferrucci (1999), queda claro que BRUTUS no puede ser creativo sino que funciona utilizando ingeniería inversa.

2.3.8. VIRTUAL STORYTELLER

Theune et al. (2003) presenta un enfoque multi-agente para la generación de historias. Para guiar la trama de la historia existe un *agente director*, el cual no podrá obligar al resto de agentes a llevar a cabo acciones, pero sí podrá influir en el entorno de los agentes. También existe un *agente narrador* que está encargado de traducir a lenguaje natural los cambios en el estado y los eventos que tienen lugar. Véase la Figura 2.9 para conocer la arquitectura de este sistema.

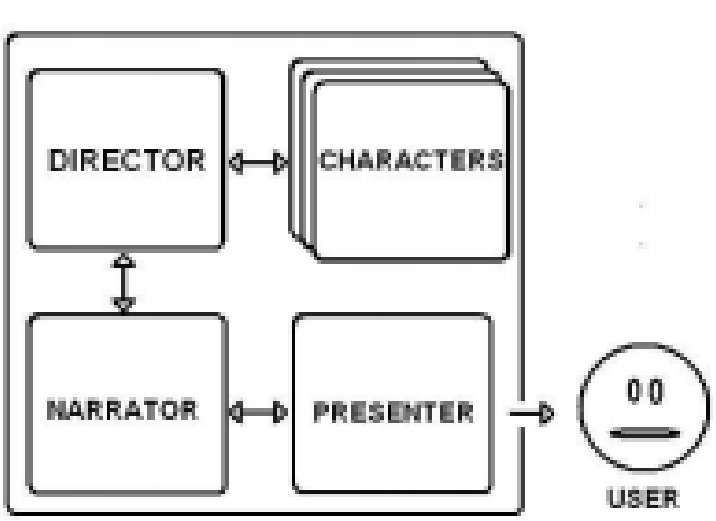


Figura 2.9: Arquitectura de VIRTUAL STORYTELLER

2.3.9. Generador de historias basado en agentes

Este fue el primer sistema narrativo que se investigó para empezar a plantear el proyecto. Se trata de un proyecto de fin de grado de unos alumnos de la Facultad de Informática de la Universidad Complutense de Madrid junto a un artículo para una conferencia en que explican brevemente el proyecto que realizaron Laclaustra et al. (2014). De esta investigación surgen las bases para empezar a desarrollar este proyecto, ya que se utiliza el mismo Sistema Multi-Agentes, JADE, y el mismo planificador para ayudar a las simulaciones. Ha sido muy importante estudiar tanto su memoria como su código fuente para poder diferenciar ambos proyectos.

Capítulo 3

Objetivos

Algunas veces hay que decidirse entre
una cosa a la que se está acostumbrado y
otra que nos gustaría conocer

Paulo Coehlo

Al disponer de un sistema base capaz de generar historias sencillas, es necesario realizar un estudio completo del mismo. El propósito de este estudio es determinar con exactitud las herramientas y posibilidades que nos ofrece el sistema y así poder planificar las mejoras necesarias sobre el mismo.

3.1. Motivaciones

A medida que se van haciendo diferentes pruebas sobre el sistema del que partimos, varias carencias y posibilidades de mejora se hacen patentes:

- Se está frente a un sistema altamente cerrado. La posibilidad de ampliar o modificar la funcionalidad del sistema se encuentra fuertemente ligada al código, impidiendo que se disponga de una forma sencilla para introducir pequeños cambios sin tener que modificar el código. Si se quiere aumentar la escala del sistema, es primordial reducir drásticamente el acoplamiento existente con el código.

- Carencia de variedad en la generación de historias. La sencillez del sistema provoca que no exista mucha diversidad en las posibles historias que se generan.

- ¿Siempre se tendrá que ver el mismo entorno de historia? En relación a lo comentado anteriormente, no es posible variar el entorno de la historia, con lo cual estamos obligados a ver que todas las historias ocurren en el mismo entorno (medieval/fantástico).

- ¿Generar más contenido sobre lo mismo ó generar diferentes tipos de historias sencillas? Son algunas de las primeras cuestiones planteadas y sobre las cuales hubo un largo período de discusión.

Resumiendo, se quiere crear un sistema que sea capaz de generar historias. Esta aplicación debe estar basada en el paradigma de Agentes y a su vez usar un sistema de planificación que permita a los Agentes interactuar y crear la historia. Se parte de un sistema que usa todo lo anterior, pero es un sistema muy básico que principalmente genera la misma historia siempre, con un bucle en medio. Ahora se quiere un sistema que permita al usuario poder generar historias dinámicas, que posibilite cambiar el entorno de la historia, que consienta al usuario incluir nuevos elementos durante la trama, como la posibilidad de crear nuevos personajes y alterar la cantidad de personajes que van a interactuar en cada historia generada. Un sistema que se adapte a la creatividad de cada usuario.

3.2. Objetivos

El enfoque principal del proyecto trata de crear un sistema que genere una amplia diversidad de historias basándose en las interacciones que tienen los personajes que integran la trama, para lo cual el sistema debe crear simulaciones sobre las acciones que puedan ejercer los personajes de estas historias. Estas simulaciones de historia deben tener un factor de repetición bajo y así conseguir que cada historia sea totalmente distintiva, "única". Además, para poder tener mayor complicidad con el usuario/lector, éste debe disponer de una serie de herramientas que permitan crear y configurar ciertas partes de la historia, como los personajes que actúan en ella, el nombre de los lugares donde se desarrolla la historia, que tipos de enemigos quiere que aparezcan en su mundo.

Todos los personajes que se utilizan en las simulaciones son Agentes, que se tratan a través de un SMA. Cada personaje tiene unos objetivos que intenta cumplir y la narración de todo el transcurso de acciones, desde su creación hasta alcanzar su objetivo o morir en el intento, proporciona fragmentos de la historia generada. Para poder saber cuál es el mejor camino para alcanzar los objetivos y evitar que nuestra historia pierda interés se utiliza un planificador.

Los objetivos planteados para este proyecto figuran como sigue:

- La eliminación del personaje objeto dentro de los protagonistas de la historia.
- Ampliar la cantidad de personajes que interactúan en la historia. Para ello se agregarán nuevos arquetipos de personajes en la historia.
- Diseñar una interfaz gráfica que permita al usuario ver cómo se genera la historia. Esta interfaz además debe permitir al usuario interactuar con la historia, al proporcionarle la posibilidad de introducir nuevos elementos a la historia que afecten el transcurso de esta.
- Incluir elementos a la generación de la historia, como objetos que afecten a los personajes ó a sus objetivos.
- Dotar a los personajes de un sistema de características similares a las que se pueden encontrar en los juegos de rol. Esto permite diferenciar aún más a dos personajes del mismo arquetipo.
- Dar al usuario el control sobre la narrativa que usarán los personajes.
- Minimizar el nivel de acoplamiento del sistema. Si se quiere crear un sistema generador que sea altamente configurable, es necesario tener un sistema robusto pero que permita al usuario editar tantos aspectos del sistema como este desee para poder generar gran variedad de historias. Los archivos configurables por el usuario se encuentran entre:
  - La escenografía o mapa.
  - Los personajes y sus objetivos.
  - Las frases de los personajes.
  - Las razas y sus atributos.
  - Los objetos de la historia.
- Tener un sistema que permita generar variedad de historias bajo diferentes entornos mediante la alteración del mismo por parte del usuario. Esto es ofrecer al usuario la posibilidad de generar historias de personajes en el espacio, en el salvaje oeste o en Gotham.

Capítulo 4

Primeros Pasos

El objetivo de este capítulo es explicar la experiencia inicial, de manera que el lector de la memoria observe el problema que surgió y como no nos quedó mas remedio que empezar de cero para poder completar correctamente la investigación.

Al empezar el proyecto, lo primero que se debía hacer era entender de donde se partía y ver si desde esa posición se podían alcanzar las cotas propuestas en la investigación, explicadas en el Capítulo 3. También es importante indicar que estas pruebas se realizan sobre la investigación sobre la que parte el proyecto, detallada en el Capítulo 2.3.9.

4.1. Punto de partida

Se parte de un trabajo reducido en el que siempre se tenía un mismo inicio (El dragón secuestra a la princesa), con un nudo(el rey contrata caballero) en bucle si se cumplía una condición(dragón mata caballero) y dos desenlaces (Un caballero salva a la princesa y se convierte en héroe o el rey se quedaba sin dinero para contratar caballeros y el dragón se queda con la princesa), esto dista de la idea que tenemos para generar historias diversas y cada vez que se ejecute la aplicación genere una historia única.

4.2. Trabajando sobre el sistema previo

4.2.1. Agentes objeto, la princesa

Las primeras pruebas que se realizaron estaban encaminadas en conseguir que todos los agentes tuviesen un objetivo que intentar cumplir. En la anterior investigación, la princesa era lo que denominaremos un Agente objeto, un Agente que realmente no tiene ninguna funcionalidad. La princesa sencillamente se movía acompañada de un lugar a otro, o bien secuestrada por un dragón o bien rescatada por un caballero, y no era capaz de realizar acciones por su propia iniciativa. Como primer objetivo se propuso cambiar este agente, de manera que a veces decidiese escaparse por su propio pie cuando se cansase de esperar un rescate.

4.2.2. La figura del villano

Sin embargo, modificar un agente clave como la princesa era un objetivo demasiado ambicioso, por lo que se busca un problema más sencillo que a su vez cumpla con el objetivo del agente con múltiples objetivos. El personaje que mejor se amolda a esta búsqueda es el caballero. En la anterior investigación el caballero solo tenia una funcionalidad, rescatar a la princesa, y si fallaba sencillamente se generaba otro caballero que ocupase su lugar. Para cambiar esto, se propone que no todos los caballeros traten de salvar a la princesa y dejarla con el rey, surgiendo así el concepto del villano. El villano es basicamente un caballero como otro cualquiera, que cuando derrota al dragón en vez de salvar a la princesa decide llevar acabo una acción malvada, como por ejemplo volver a secuestrarla, provocando que otro caballero deba venir a salvarla. Así se aumentan la cantidad de crear historias distintas con una idea sencilla.

4.2.3. Un sistema cerrado

Sin embargo, estos cambios no resultaban tal y como estaba desarrollado el sistema anterior. El planificador no soportaba tener que elegir aleatoriamente entre dos caballeros cual va a tratar de ser héroe y cual va a tratar de ser villano y, bajo el entendimiento del código disponible, no se podía crear un caballero de manera sencilla que en el punto de inflexión para salvar a la princesa decidiese entre rescatarla o volverla a raptar.

Como primera solución, se propone la creación del personaje `Villano`, de manera que cuando se crea un caballero, se elija de manera aleatoria si este será `Caballero` o `Villano`, de manera que llegado el momento de tratar a la princesa lo harían de manera distinta. Sin embargo la aplicación no estaba pensada para soportar esto, debido a que las batallas debían ser entre un dragón y un caballero, le costaba adaptar el rol del caballero para que fuese

dragón por lo que el planificador no se entendía con los cambios, entre otros factores.

A este problema había que sumar la nula adaptabilidad de la aplicación para adaptarse a nuevos entornos, la creación de una interfaz gráfica que interactúe de manera correcta con el código, tener que modificar los personajes a sus nuevas cualidades... Así que llegados a este punto, se decide replantear la aplicación desde cero.

4.3. Creando un nuevo sistema

Al replantear la aplicación desde cero hay que tener en cuenta que se parte de la base de la anterior investigación, reutilizando cierta parte del código que aún es funcional para este proyecto. Sin embargo, se ha de adaptar a los personajes y su manera de actuar a las nuevas necesidades, así como rehacer el código de una manera mas estructurada, que se consigue respetando correctamente el las tecnologías explicadas en el Capítulo 2, y minimizando el nivel de acoplamiento del sistema para que sea mas adaptable, lo cual también facilitará el uso de esta investigación en futuros proyectos.

Ahora que ya se está donde se quiere es el momento de sacar la ambición, ya que se ha eliminado una restricción y se puede elaborar un proyecto vacío incluyendo una base que respalda la idea del proyecto y permite avanzar en la investigación con paso seguro.

Es en este punto donde se propone gran parte del diseño, como la creación de nuevos personajes, nuevos antagonistas, e incluso poner algunos personajes totalmente secundarios que sencillamente le den un toque picante a la historia. De la misma manera, se empieza a trabajar en la idea de una aplicación altamente configurable y en una interfaz gráfica que interactúe correctamente con la aplicación.

El análisis y la creación del código se discutirá con mas detalle en los siguientes capítulos, donde se relata detalladamente como han sido diseñados e implementados todos los aspectos de la aplicación durante la realización del proyecto.

Capítulo 5

Arquitectura del generador de historias

Comentar el código es como limpiar el cuarto de baño; nadie quiere hacerlo, pero el resultado es siempre una experiencia más agradable para uno mismo y sus invitados

Ryan Campbell

Al momento de empezar a crear una historia podemos plantearnos diferentes enfoques con el fin de abordar esta cuestión de la manera mas sencilla posible. El aspecto sobre el que van a estar centradas las historias generadas son los personajes. Ya que serán estos quienes al momento de interactuar entre sí generarán la historia que se desea.

Para hacer que existan los personajes dentro del sistema es necesario definirlos y darles las herramientas necesarias para que estos puedan interactuar. Dicho esto, los personajes deben existir dentro del Sistema Multi-Agente. El SMA es el componente capaz de dotar a los personajes de las herramientas que les permiten enviar mensajes para comunicarse con otros personajes y existir como agentes software. Usando JADE[1], este sistema permite controlar la existencia de los agentes y poder manipularlos como tal.

El siguiente enfoque tiene más que ver con el aspecto del entorno de la historia. Tener varios personajes en la historia no implica nada si estos no disponen de un entorno sobre el cual desarrollar sus acciones y vivencias. Es necesario ahora otorgar al sistema un poco más de profundidad en el asunto, por lo que se añade un mundo.

El Mundo es un componente que permite a los personajes disponer de un nuevo espacio, un lugar donde queden reflejadas sus acciones. Sin la exis-

[1] http://jade.tilab.com/

tencia del Mundo se podría decir que los personajes viven en la nada. Visto de otra forma, es como si todos estuviesen en una habitación vacía, en un espacio unidimensional en el cual no tendría sentido realizar acciones como moverse, negociar con terceros personajes, dudar, etcétera. La existencia del Mundo como nueva dimensión espacial aporta a los personajes la capacidad de existir en un plano. Los personajes dejan de conocer la situación de otros y llegan a la necesidad de interactuar con su entorno para poder realizar acciones relevantes en una historia.

Ahora que ya se tiene un ecosistema creado con los personajes y su entorno, se necesita de un sistema capaz de otorgar la lógica e inteligencia a los personajes, dicho sistema es el planificador. El planificador es el componente que permite a los personajes "pensar" en la secuencia de acciones que deben realizar para poder cumplir sus objetivos dentro de la historia. Estas acciones ayudarán al personaje desde decidir a cual dirección moverse hasta en qué momento pelear con otro personaje.

Descrito el núcleo importante del sistema, se han juntado los componentes mínimos para poder generar una historia. Aún así el usuario no está incluido dentro de todo esto y no tiene sentido tener un sistema que trabaje para sí en lugar de dejarle a un usuario una copia de los resultados obtenidos.

Es necesario el uso de algún componente o sistema que nos permita recoger el flujo de historia final. Saber cuáles han sido los mensajes que intercambiaron los personajes y las acciones que realizaron. Usar un *logger* permite guardar un registro del momento en el que ocurre cada eventos de la historia.

Sin embargo, al ser un sistema pensado para ser utilizado por un usuario, falta disponer de una capa que permita a este interactuar con el sistema. La interfaz gráfica otorga al usuario la capacidad de visualizar información adicional durante el momento de la simulación. Es mediante esta capa, que el usuario puede también añadir nuevos elementos a la generación de historias.

A continuación hablaremos con más detalle sobre cada uno de estos componentes y sobre otras partes del sistema necesarias o que se ven afectadas por estos.

Para tener una referencia que permita visualizar de manera sencilla la arquitectura del proyecto se ha incluido la Figura 5.1.

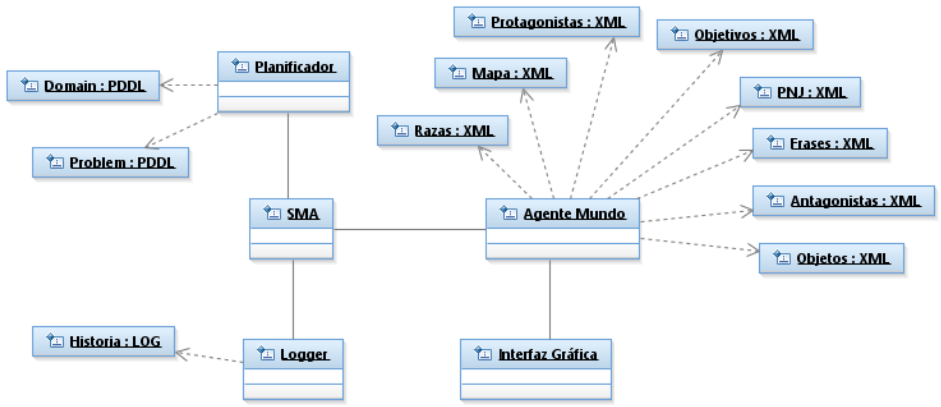


Figura 5.1: Arquitectura del proyecto

5.1. Personajes y SMA

En el Capítulo 2.1 se habla extensamente de las partes principales que componen al SMA de la aplicación, JADE y la plataforma que nos ofrece FIPA.

Cada uno de los personajes es implementado como agente software usando la plataforma JADE. Esto permite que los personajes puedan usar los estándares definidos por FIPA e interactuar con otros personajes.

Cada Agente del sistema es un hilo de ejecución independiente dentro de este. Esto permite que cada uno de los personajes pueda existir y actuar independientemente del resto de agentes haciendo uso de los diferentes *behaviours* JADE. La unión de estas herramientas compone el núcleo SMA.

5.2. El Agente Mundo

El Agente Mundo tiene la función de servir como una plataforma en la cual los personajes se apoyan para poder desarrollar su funcionalidad. Más allá de sólo encargarse de la interacción de los personajes con la escenografía, el Mundo es el encargado de gestionar los siguientes aspectos:

- Activar y gestionar las llamadas al SCD (*Sistema de Carga de Datos*). Ver en la Sección 5.4.
- Actualizar el estado actual de cada personaje.
- Gestionar ciertos eventos entre personajes.
- Controlar las acciones tomadas contra la interfaz gráfica. Véase más adelante la Sección 5.5.

- Proporcionar a los personajes los archivos necesarios para poder usar el planificador(Sección 5.3).

Aunque la implementación del Mundo se hace como un Agente[2], debido a la naturaleza de éste no se considera su pertenencia al SMA.

Respecto a la interacción con la interfaz, el Agente Mundo funciona como el controlador al cual la vista envía la información recibida por el usuario y se encarga de validar ésta.

Al ser un Agente, la forma en la que gestiona la mayoría de funciones mencionadas anteriormente es a través de comportamientos JADE. Específicamente usando el tipo CyclicBehaviour. El motivo por el que todos los comportamientos que debe usar este Agente sean cíclicos es que principalmente este Agente que debe estar en una escucha continua de peticiones por parte de todos los demás Agentes. Por tanto, modelar el funcionamiento de este Agente debe hacerse mediante un conjunto de comportamientos que estén constantemente evaluando si se han recibido peticiones nuevas. Todo esto se puede ver la arquitectura de clases del Agente Mundo (ver Figura 5.2).

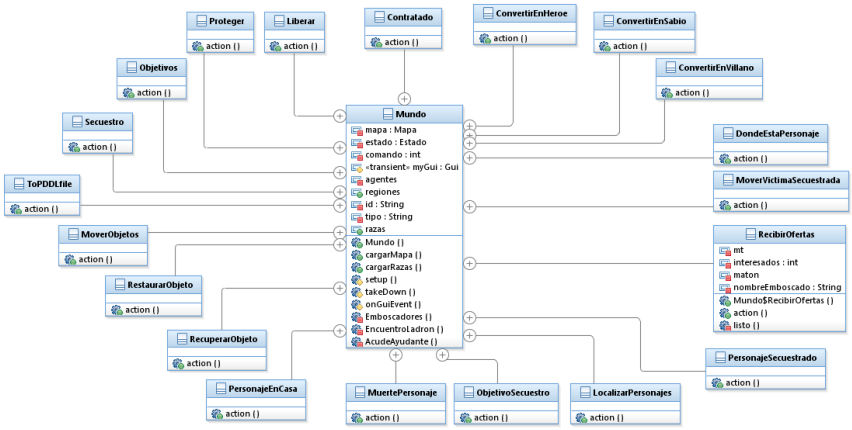


Figura 5.2: Diagrama de Clase del Agente Mundo

5.3. El Planificador

El planificador es el componente del sistema que se encarga de aportar "inteligencia" a los personajes. Recordar que el propósito de los personajes es resolver un problema y la historia general contará lo que ha hecho cada personaje para poder dar solución a sus problemas.

[2]Este es un agente especial del tipo GUIAgent, lo que permite la interacción con una interfaz gráfica. Más información en Bellifemine et al. (2007).

La solución a un problema u objetivo dado al planificador vendrá dada al procesar la información que éste recibe mediante los ficheros de dominio y problema (explicados en el Capítulo 2.2.3).

Para la generación de historias, el fichero de dominio va a contener el tipo de elementos que existen dentro de estas, las acciones que serán capaces de realizar cada uno de los personajes y alguna otra información clave que indique al planificador cómo actuar. Todo esto significa que es en este fichero donde realmente se le da forma al tipo historia que se desea generar, se definen las características del universo de la historia.

Aquí se presenta gran parte de la complejidad al momento de definir una base para poder generar historias. La calidad y riqueza de cada historia se ve afectada implícitamente por la variedad, complejidad y el nivel de abstracción con el que se programe el dominio. Para tener una idea de todo esto, podemos imaginar que implicaría la creación un *universo* con las siguientes restricciones:

- Sólo existen personas, objetos y habitaciones.
- Una persona es capaz de cambiar de habitación.
- Una persona es capaz de recoger un objeto.
- Una persona es capaz de soltar un objeto.

Definir un universo tan simple acota la cantidad de permutaciones que se pueden realizar con las acciones, y básicamente se habrá creado un universo en el que las personas sólo pueden transportar objetos de una habitación a otra.

Por el contrario, cuanto más se profundiza en las características deseadas dentro del universo que se quiere crear, por ejemplo definiendo aspectos cómo:

- Mover pie/brazo derecho/izquierdo, ojo derecho/izquierdo.
- Saltar obstáculo pequeño, esquivar obstáculo grande.
- Personas, animales, insectos, objetos, paisajes.
- Temperatura, altitud, latitud.
- Color del pelo, piel y ojos.
- Personalidad.

Se amplían las herramientas que se pueden usar para dar vida a los personajes y definir una mayor variedad de comportamientos para estos o su entorno si así se desea.

Hay que tener en cuenta que realizar una acción implica el cumplimiento de una serie de precondiciones y efectos especificados en función de la acción. La sucesión de un conjunto de acciones en cierto orden deben asegurar que se consiguen los efectos que hacen cumplir un objetivo dado. Nótese que esto también conlleva un aumento considerable de la complejidad de los problemas a resolver y al tiempo que tarda el planificador en poder encontrar una solución plausible a los problemas propuestos.

Por otra parte, el llamado fichero del problema contiene la información del estado actual de las características definidas en el dominio que cada personaje conoce y que, basándose en el estado de estas, sabe si existe una solución al problema que intenta resolver. Usando el ejemplo del universo simple que se comentó anteriormente, el fichero del problema contendría por ejemplo la siguiente información:

- La habitación **A** está junto a la **B**.
- El objeto **X** se encuentra en la habitación **A**.
- La persona **P** está en la habitación **B**.
- Objetivo de P: *el objeto X debe estar en la habitación B.*

Cuando el planificador recibe este fichero problema, lo que hace es buscar en el dominio para saber cuáles son las acciones que *puede* realizar la persona para lograr su objetivo. Una vez evaluadas las diferentes posibilidades, el planificador devuelve el plan de acción (la planificación) que debe llevar a cabo el personaje en estricto orden para conseguir que su objetivo se cumpla. La salida que devolvería el planificador en este caso sería algo como lo siguiente:

1. Mover P a la habitación A.
2. Recoger el objeto X.
3. Mover P a la habitación B.
4. Soltar el objeto X.

La ejecución ordenada de la planificación devuelta por el planificador asegura al personaje que logra conseguir con éxito su objetivo. Los cuadros de Código 5.1 y 5.2[3] muestran un ejemplo sencillo de cómo se estructura cada fichero y cómo es la sintaxis de PDDL.

[3]Recuperados de Bellifemine et al. (2007)

Código 5.1: Ejemplo de fichero domain.pddl

```
(define (domain briefcase-world)
   (:requirements :strips :equality :typing :conditional-effects)
   (:types location physob)
   (:constants (B - physob))
   (:predicates
      (at ?x - physob ?l - location)
      (in ?x ?y - physob)
   )
   (:action take-out)
      :parameters (?x - physob)
      :precondition (not (= ?x B))
      :effect (not (in ?x))
   )
   ...
)
```

Código 5.2: Ejemplo de fichero problema.pddl

```
(define (problem get-paid)
   (:domain briefcase-world)
   (:init (place home) (place office)
   (object p) (object d) (object b)
   (at B home) (at P home) (at D home) (in P))
   (:goal (and (at B office) (at D office) (at P home)) )
)
```

5.4. Carga de Datos y Archivos de Configuración

Además de los archivos pddl usados para la planificación, crear un sistema capaz de permitir al usuario alterar la mayor cantidad de aspectos posible al momento de generar nuevas historias, exige que cierta información del sistema sea configurable fuera de la compilación.

Esta funcionalidad es la que permite al usuario indicar al sistema cuáles son los actores y el entorno donde se van a generar las diferentes historias y le da la libertad de cambiar los diferentes aspectos que éste hubiese planteado.

Con vistas en esto, el sistema permite al usuario definir y modificar con alta libertad los siguientes aspectos:

- Escenografía.
- Personajes protagonistas.
- Personajes antagonistas.
- Frases de los personajes.
- Objetivos que han de cumplir los personajes.

- Objetos de la historia.
- Razas y atributos.
- Personajes no jugadores (PNJ's).

Estos ficheros están codificados en XML[4]. Este es un lenguaje que nos permite diseñar con mayor simpleza el esquema de los datos que el usuario puede modificar en cada uno de los aspectos que se han mencionado anteriormente. Algunas restricciones, recomendaciones y comentarios generales que influyen en todos los archivos son:

- Los atributos *nombreX* deben ser atómicos; una sola palabra.
- Los valores numéricos en los atributos, salvo para la vida, mayormente deben estar comprendidos entre 0 y 10.
- Es importante la distinción entre mayúsculas y minúsculas para los nombres que se elijan puesto que el sistema en algunos puntos hace distinción de caracteres, esto producirá un error en el sistema.
- Algunos atributos de las etiquetas XML necesariamente tienen que tomar un valor de entre ciertos propuestos.
- Aunque se encuentre en dos ficheros diferentes, un mismo nombre es único para todos, por ejemplo en el caso de las localizaciones.

El Sistema de Carga de Datos (SCD de ahora en adelante) que se encarga de cargar la información de cada fichero de configuración en el sistema tiene una serie de restricciones específicas para cada uno de estos que se verán con más detalle en los capítulos correspondientes. La arquitectura del SCD se puede ver en la Figura 5.3.

[4] eXtensible Markup Language

Figura 5.3: Diagrama de Clase de los SCD

5.5. Interfaz Gráfica del Sistema

La inclusión de un módulo que aporta un componente visual al sistema ayuda a incorporar ciertas funcionalidades de una manera sencilla a éste mucho más allá del factor estético.

La Figura 5.4 muestra la pantalla principal de la interfaz. Los mensajes producidos por los agentes se muestran en esta pantalla para ir creando el hilo de historia.

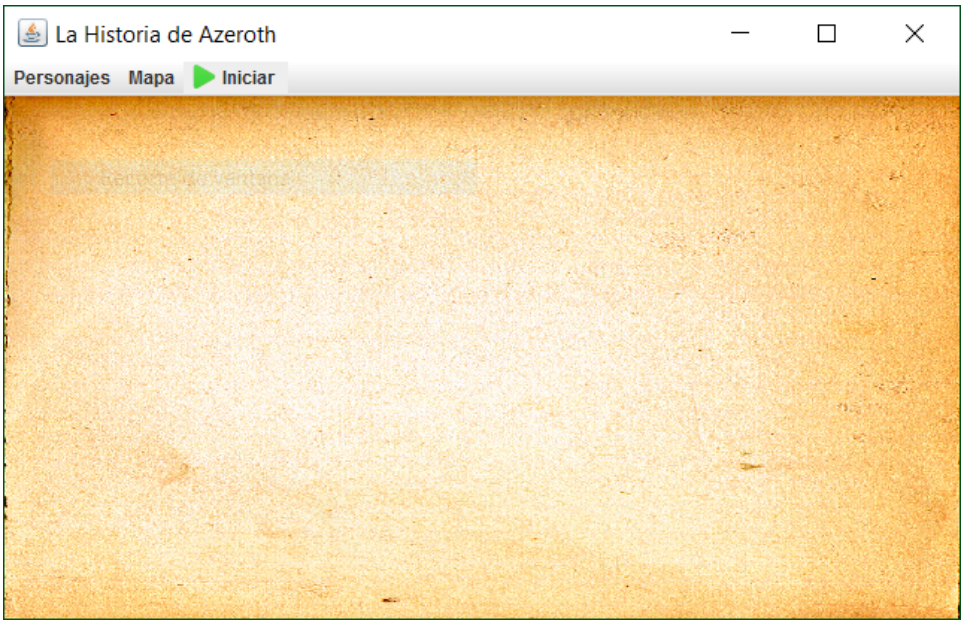


Figura 5.4: Pantalla principal de la interfaz

La barra de menús contiene tres botones que nos permiten interactuar con el sistema:

- Personajes: Ofrece cuadros de herramientas que permiten incorporar nuevos personajes a la historia.
- Mapa: Permite seleccionar diferentes imágenes referentes a la escenografía (estas imágenes podemos verlas en el Capítulo 6).
- Iniciar: Botón que empieza la ejecución de los personajes incorporados usando el cuadro de herramientas.

La Figura 5.5 es un ejemplo del cuadro de herramientas que permite al usuario agregar nuevos personajes a la historia mientras ésta se está desarrollando. El usuario puede ir añadiendo personajes continuamente en cualquier momento del transcurso de la historia y todos estos sólo empezarán a ejecutarse cuando se pulse el botón de Iniciar de la pantalla principal. Esto se hace así para dar algo de margen y mantener cierto control a la hora de empezar la ejecución de personajes nuevos.

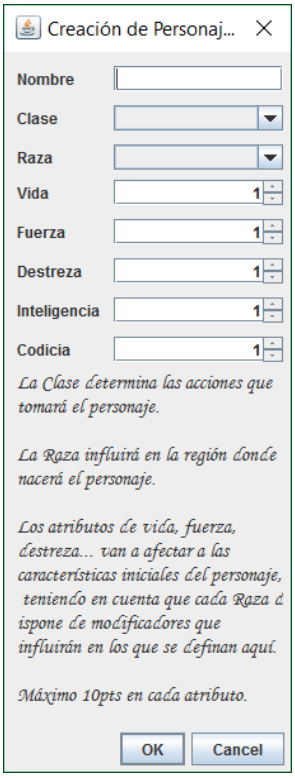


Figura 5.5: Herramienta para incorporar nuevos personajes

Todo la interfaz mostrada está soportado por un sistema de clases que queda reflejado en la Figura 5.6.

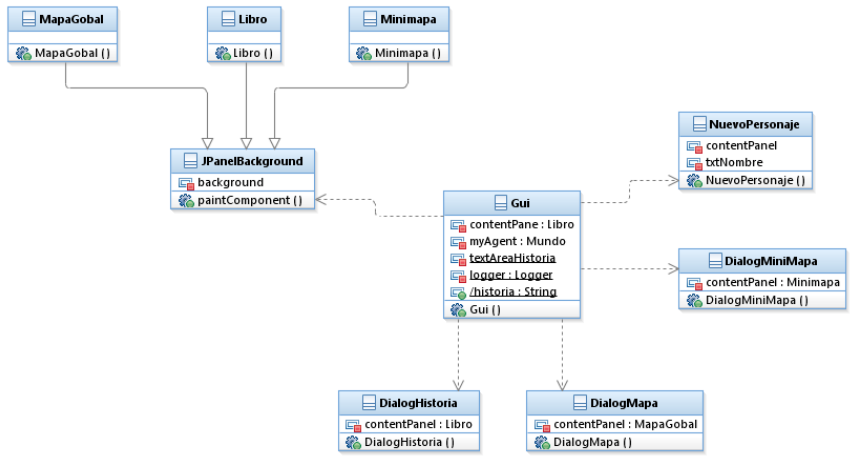


Figura 5.6: Diagrama de Clase de la GUI

5.6. El Logger

Aunque la interfaz gráfica se encarga de mostrar al usuario en momento de ejecución todas las acciones que han llevado a cabo los agentes y que terminan generando la historia, tener un recopilatorio ó registro de todos eventos y las simulaciones realizadas requiere de una parte del sistema que se encargue de registrar todo lo sucedido en un fichero.

Este componente registra toda la información que generan los agentes, incluyendo el emisor del mensaje, el momento de emisión y el mensaje en sí. No obstante, existen ciertos mensajes que no son registrados en el logger ya que restarían claridad al archivo, estos son los mensajes de control que usan los agentes para comunicarse entre ellos, como los vistos antes en la sección 2.1.5.

Son variados y considerablemente complejos cada uno de los componentes que integran toda la arquitectura del sistema que se está desarrollando. Los capítulos siguientes tratan de explicar con mayor nivel de detalle las características que tienen otros elementos internos de la arquitectura y sobre cómo son implementados. Por último se invita al lector, si está interesado en conocer aspectos más específicos sobre el modelo del sistema, a revisar la sección de apéndices al final de este documento donde se incluyen algunos ejemplos de secuencia de los mensajes FIPA transmitidos entre los agentes durante una simulación.

Capítulo 6

El Mapa

Ni el más sabio conoce el fin de todos los caminos.

Gandalf en La Comunidad del anillo por J.R.R.Tolkien

Las mejores historias siempre se caracterizan por un mapa detrás que les da soporte, que compone los cimientos del mundo que el autor quiere introducir. Tratando de dotar a las historias de un mapa así de consistente, se decide crear un mapa que se adecue a un mínimo de necesidades, este mapa dotará a la historia de individualidad y servirá para orientar al usuario, de manera que este sepa donde se produce la acción.

6.1. Diseño del Mapa

Cuando se plantea el nuevo concepto de mapa lo primero que se decide es enfocarle dentro de un ámbito fantástico y crear un mundo al viejo estilo de las novelas fantásticas, como las de Tolkien, con nombres específicos que representen reinos y civilizaciones para darle un aire más personal a las historias, pero que a la vez sea lo suficientemente claro y no este sobrecargado de información nueva para el usuario.

6.1.1. Punto de partida

Cuando se empieza a diseñar el mapa se decide partir del mapa del proyecto Laclaustra et al. (2014), debido a que este mapa es un plano muy sencillo, véase en la Figura 6.1, y permite trabajar en un entorno controlado.

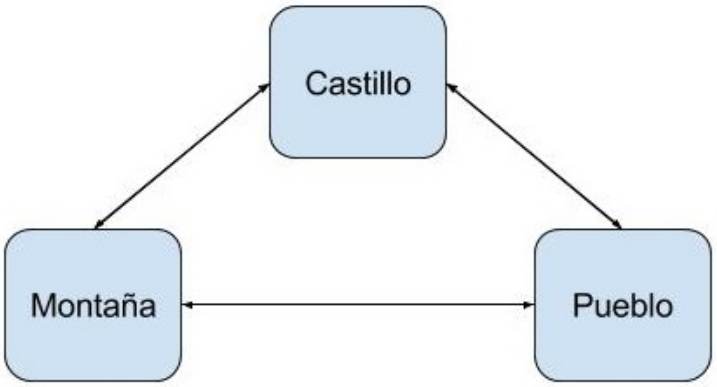


Figura 6.1: Mapa Base

Las primeras pruebas consisten en trabajar sobre el planificador, consultable en el Capítulo 5.3, para ver como éste lidia con un problema y trata de resolverlo con respecto a un mapa, de manera que cuando le se presenten nuevos mapas se pueda mantener el control sobre el comportamiento de la historia. Con está idea en mente, se desconecta la montaña del pueblo y se interpone entre el castillo y la montaña dos nuevas localizaciones, como se ve en la Figura 6.2, el bosque y el lago. Con esto se logra ver que el planificador devuelve una ruta aleatoria si se quiere ir del castillo a la montaña o viceversa, ya que los agentes tienen dos posibles caminos por los que llegar a su destino.

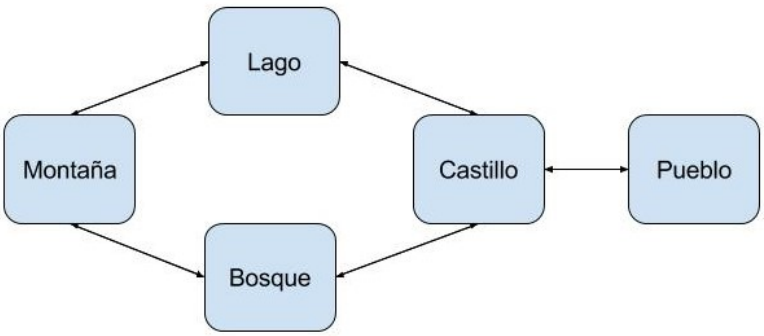


Figura 6.2: Mapa de primeras pruebas

La siguiente prueba que se realiza al planificador es para saber como se escogen las localizaciones por las cuales transcurren los Agentes para llegar a su destino. Tras diversas pruebas con distintas localizaciones y algunas trampas, se descubre que el planificador siempre trata de escoger el camino más corto posible para cada Agente. Si hay dos iguales, elige indistintamente entre estos e intenta no repetir con el mismo Agente un mismo lugar, es decir, observando la Figura 6.2 si un Agente ha pasado por el bosque elige, en la mayoría de pruebas, volver por el lago y viceversa. Esto permite conocer en profundidad la manera de actuar del planificador y así se pueden establecer los límites que ha de tener cada mapa que se vaya a utilizar. Una vez conocidos estos límites y como los trata el planificador, se empieza a desarrollar el nuevo mapa.

6.1.2. Creando el mapa

Al empezar a desarrollar el nuevo mapa, primero se plantea el mismo geográficamente, de manera que un pequeño mapa vaya tomando forma para dar origen a uno más grande. Para poder organizarlo bien, se utilizan varios conceptos para dividirlo. Así, se crean las regiones, que representan estados subdivididos a su vez en localizaciones, que conforman los lugares de renombre del propio estado, como ciudades, bosques o pueblos.

Para desarrollar el mapa, se crea un archipiélago de islas donde tres islas pequeñas coronan a una isla mayor que a su vez se divide en otras tres partes, dividiendo así el mundo en seis regiones, que representan las seis divisiones en las que esta fraccionado el mapa consultable en la Figura 6.3, donde cada región tiene un número diferente de localizaciones, que representan territorios más específicos por los que realmente se moverán los personajes. También se dota a cada una de estas regiones de un nombre en específico para distinguirlas entre ellas y entender mejor el mapa.

Figura 6.3: Mini Mapa Final

Con ánimo de seguir aumentando el mapa, se procede a profundizar en el contenido de las regiones. Con el objetivo de que cada región sea única y no se tengan sencillamente seis zonas con seis nombres distintos, se decide proporcionar a cada una, uno o varios elementos distintivos que la diferencien del resto de regiones a través de localizaciones, siendo algunas de estas exclusivas de cada región. Así, por ejemplo, se decide que solo en dos regiones habrá un castillo, limitando así el número de reinos existentes en la historia a dos.

De la misma manera, se forman localizaciones particulares en ciertas regiones. Por ejemplo, en una de ellas estará la herrería, en otra la taberna, en otra el hospital. Estas localizaciones cumplen las veces de sitios sencillos que albergan personajes secundarios que pueden cambiar el curso de la historia o sencillamente hacer acto de presencia en él. Así en estos sitios, aguardan estos nuevos personajes esperando el momento de su intervención, estando estos personajes ligados a una localización particular en específico, como en este caso serán el herrero, el tabernero o la amable enfermera.

Con el fin de dotar al resto de personajes de lugares en los que entrar en la historia, se decide añadir una nueva característica al mapa, concretamente la de tipos de localizaciones. A través de esta nueva característica, se busca clasificar las localizaciones según un elemento común. Así, todas aquellas

localizaciones en el mapa que sean bosques podrán ser tratados como tal para poder trabajar correctamente con ellos en conjunto. De esta manera se clasifica a las localizaciones en los siguientes tipos: castillo, pueblo, urbano (que se refiere a las localizaciones específicas previamente comentadas como la herrería), bosque, lago, guarida y camino.

Esta clasificación en tipos ayudará a la hora de crear personajes, asignándole a cada protagonista o antagonista un tipo de localización en concreto, de forma que al crearse dicho personaje sea situado aleatoriamente en una de las localizaciones que tengan el tipo al que dicho personaje pertenece. De esta manera, si al protagonista o antagonista X le asignas el tipo de localización Y, a ese personaje se le cede una localización específica Z que pertenece al conjunto de localizaciones del tipo Y. El orden final se puede consultar en la Tabla 6.1:

Protagonista o Antagonista	Tipo de Localización
Allegado a la Víctima	Castillo
Víctima	Castillo (el del Allegado)
Aspirante a Héroe	Pueblo
Ayudante del Héroe	Urbano
PNJ's	Urbano(elección específica)
Secuestrador	Guarida
Emboscador	Bosque
Ladron	Lago
Asesino	Cualquiera

Tabla 6.1: Tabla de Relación Clase-Localización

Finalmente, se decide que todas estas características puedan ser configuradas por el usuario, a través de un archivo XML, donde el usuario pueda diseñar, siempre dentro de unos ciertos parámetros, su propio mapa.

Una vez ya decidida la funcionalidad que tendrá el mapa, es el momento de plasmarlo en un mapa virtual, una imagen intuitiva que ayude al usuario a situarse en la historia. Para ello, se dibuja un primer boceto en papel y se digitaliza, a través de la herramienta GIMP. Una vez correctamente digitalizado se introduce en la interfaz gráfica, aportando un gran impacto visual y quedando listo para ser consultado (ver Figura 6.4).

Figura 6.4: Mapa Final

6.2. Implementación del Mapa

Para que el mapa sea configurable y pueda cargar sus datos a través de un archivo XML es necesario un sistema de carga e interpretación que conlleva consigo distintas aportaciones al código, que a continuación se ven en profundidad.

De igual manera, para poder usar de forma correcta el mapa, es importante destacar la necesidad de el `Agente Mundo`, que puede cargar e interpretar en toda su magnitud el mapa, tal y como se explica en el apartado 5.2, lo que precisa de la adición de las clases que permiten que esto ocurra. La arquitectura que sigue esta implementación es consultable en el diagrama de clases de la Figura 6.5.

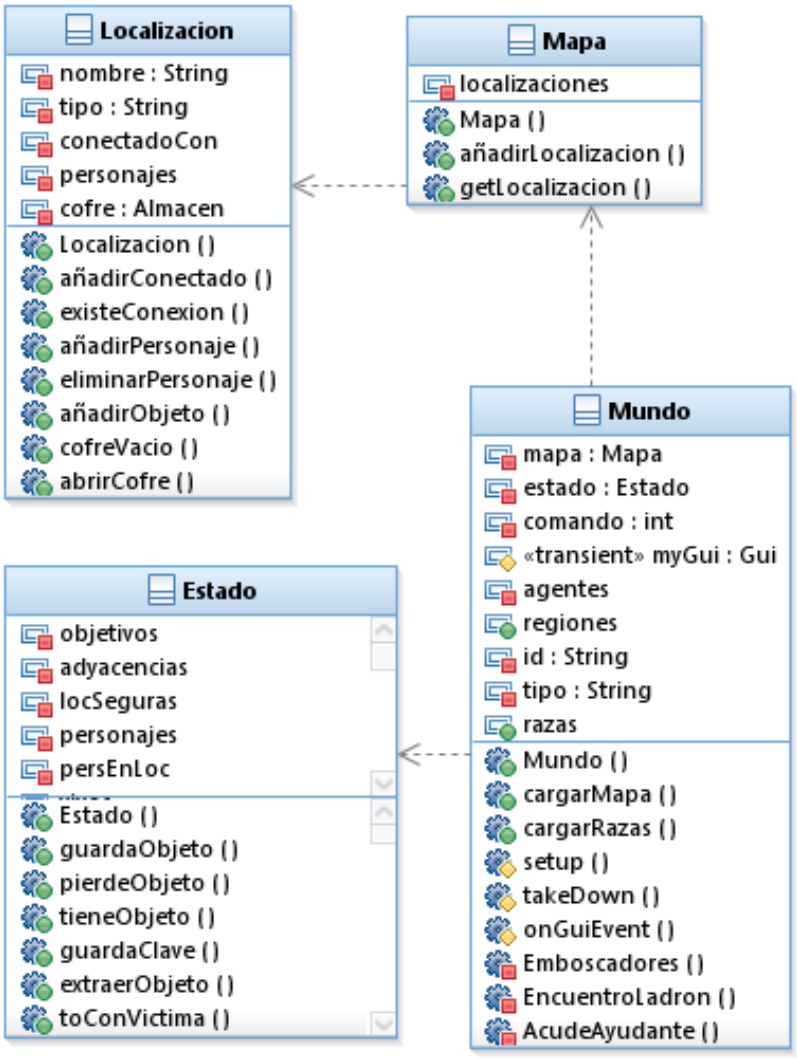


Figura 6.5: Diagrama de Clase del Mapa

6.2.1. Mapas configurables por el usuario

El mapa es un archivo XML donde la información llega en forma de *Strings*, que representan los nombres de las distintas localizaciones, y se carga directamente en la aplicación.

Para poder usarse de la manera correcta, primero se debe contar con un SCD capaz de interpretar los datos del mapa. Sin embargo, los datos del mapa deben ser conocidos por el `Agente Mundo` y por tanto se integra este SCD dentro del propio agente dándole la funcionalidad de `CargarMapa()`.

La función que actúa como el SCD del archivo del mapa funciona bá-

sicamente como el resto de SCD's, ver Capítulo 5.4, con la particularidad de que este inicia un objeto de la clase `Mapa`, que viene a ser un *ArrayList* de localizaciones, y recorre los nodos del archivo XML buscando la etiqueta `<localización>` a partir de la cual carga nuevas localizaciones. Sin embargo, al cargar el mapa no solo se cargan las localizaciones, sino que también se debe cargar la matriz de adyacencia de estas para que el `Agente Mundo` sea capaz de interpretar correctamente el mapa. Para ello, dentro de cada localización, en la carga se busca la etiqueta `<conectadoCon>` donde se reflejan aquellas localizaciones adyacentes a la localización que esta siendo cargada.

Para que estas cargas se puedan realizar correctamente es necesaria la inclusión de dos nuevas clases, la clase `Mapa` y la clase `Localización`, que se explican en detalle en el apartado 6.2.2.

Para introducir la relación entre clases y localizaciones, al archivo XML se le introduce la relación de tipos, como ya se comentó en el capítulo de diseño. Estas localizaciones se guardan en un *ArrayList* auxiliar, donde están ordenadas según su tipo, permitiendo así a los personajes acceder al tipo que buscan y decidir de manera aleatoria su localización de origen entre las localizaciones de dicho tipo.

El problema que representa esta implementación es que al necesitar el mapa de una configuración tan específica, el archivo XML debe crearse bajo unos parámetros determinados, pues si no, la aplicación no funcionará de una manera correcta. Para que esto no ocurra, en el archivo XML del mapa se incluye una pequeña guía de cómo construirlo, aparte de proponer por defecto el mapa final cuya figura aparece en el diseño, que también hace las veces de ejemplo. El formato de construcción se puede consultar en el cuadro de Código 6.1.

Código 6.1: Formato XML del Mapa

```
<?xml version="1.0" encoding="UTF-8"?>
<Mapa>
   <region nombre="[nombreRegión]">
      <localizacion id="[nombreLocalización]"
          tipo="[Pueblo|Bosque|Guarida|Urbano|Castillo|Lago]" >

          <conectadoCon>[nombreLocalización [nombreLocalización ...]]
             </conectadoCon>
       </localizacion>
   </region>
</Mapa>
```

6.2.2. Las Clases Mapa y Localizacion y cómo localizar personajes

La Clase `Localización` permite trabajar con las localizaciones de una manera más funcional, ligando al *String* identificativo de la localización con su tipo (bosque, lago, etc.)

De igual manera, la Clase `Localizacion` permite al `Agente Mundo` conocer que personajes y objetos interactúan con ella, así como con que localizaciones está conectada.

La Clase `Mapa` posibilita tener un conjunto de localizaciones fácilmente accesible por el `Agente Mundo` y acceder a localizaciones específicas de este.

De forma similar, para localizar a los personajes se crea la Interfaz `Vocabulario` junto al Enumerado `Mitologia`, que permite ligar tipos de personajes a tipos de localizaciones específicas para posteriormente elegir, mediante una selección aleatoria dentro de la lista de localizaciones específicas de cada tipo de personaje, cual será la localización inicial del personaje.

Capítulo 7

Los Personajes

Si quieres dar variedad a tu historia esta debe contar con una gran diversidad de personajes, cada uno con sus virtudes y limitaciones, y a cuantos más personajes haya más posibilidades tendrás de crear diferentes historias, ya que cada uno podrá actuar de manera distinta en cada ejecución.

Con este objetivo en mente se propone un nuevo diseño para los personajes. Sin embargo las posibles historias que se pueden generar comparten siempre una misma estructura, ya que no puede ser de otra manera. Siempre aparecerá un antagonista que logrará secuestrar a la víctima, el allegado a esta víctima pedirá ayuda a los distintos aspirantes a héroes y estos acudirán a la llamada intentando salvarla, a partir de lo cual irán apareciendo todos los personajes incluidos en la obra. Para poder diseñar correctamente estos diversos personajes lo primero que se hace es concretar unos tipos que definan en cierta medida cual será la manera de participar en la historia. De esta manera se decide dividir a los posibles personajes en tres bloques concretos, que son: Protagonistas, Antagonistas y Personajes no Jugadores(PNJ's).

De esta manera los Protagonistas son los encargados de llevar el peso de la historia, representando aquellos personajes con los que el usuario se puede identificar y asumen el rol de "buenos" dentro de la narración. Los Antagonistas, por su parte, actúan en contra de los intereses de los Protagonistas, proponiendo problemas que estos deben resolver para el transcurso de la historia y asumiendo el papel de "malos" en la misma. Por último, los PNJ's representan los personajes secundarios que intervienen en una narración, modificando la manera en la que esta discurre con un papel de apoyo u oposición con respecto a los Protagonistas.

También es importante indicar que todos los personajes se planifican en algún momento de su ejecución. Para entender realmente como se planifi-

can y el papel que juega el planificador en su ejecución, es recomendable comprender bien el Capítulo 5.3.

7.1. La Clase Personaje

Antes de empezar a considerar lo que cada personaje en específico es capaz de hacer y que función tiene pensada para el desarrollo de la historia, es importante investigar la clase que modela el funcionamiento de los personajes.

La clase `Personaje` representa un papel fundamental en la implementación de la aplicación. En concreto, la clase `Personaje` proporciona una superclase a partir de la cual el resto de clases se desarrollan. Para ello, cuenta con diferentes métodos y atributos comunes a los distintos tipos de personajes. Hay que resaltar que los PNJ's, al tratarse de personajes más planos, aunque pueden realizar todas estas acciones, habrá algunas que no realizarán en el transcurso natural de la historia, como por ejemplo planificarse.

Gracias a esta clase, todos los agentes que representan personajes comparten ciertos métodos. Estos métodos les permiten realizar tareas comunes a todos ellos, como cargar la información relativa al mundo, localizarse en el mapa o moverse a través de él. También es importante destacar a los métodos `planificar` y `mandarCrearArchivo` que permiten a los distintos personajes planificarse y crear sus archivos `PDDL`, siendo una parte vital de la aplicación ya que esto les permite interactuar con el planificador(véase el capítulo 5.3). Aparte, todos los personajes comparten ciertos atributos comunes, como pueden ser sus puntos de vida, su sexo o su localización, cuyos respectivos métodos `getters` y `setters` también están incluidos en esta clase para poder ser usados correctamente.

Por lo tanto gracias a esta clase, se puede unificar el código de una manera más sencilla, evitando la redundancia de código y proponiendo una superclase común sobre un elemento que así lo requiere, provocando que la aplicación sea mas intuitiva y fácil de desarrollar. Para visualizar mejor esta arquitectura, se puede consultar su correspondiente diagrama de clases en la Figura 7.1.

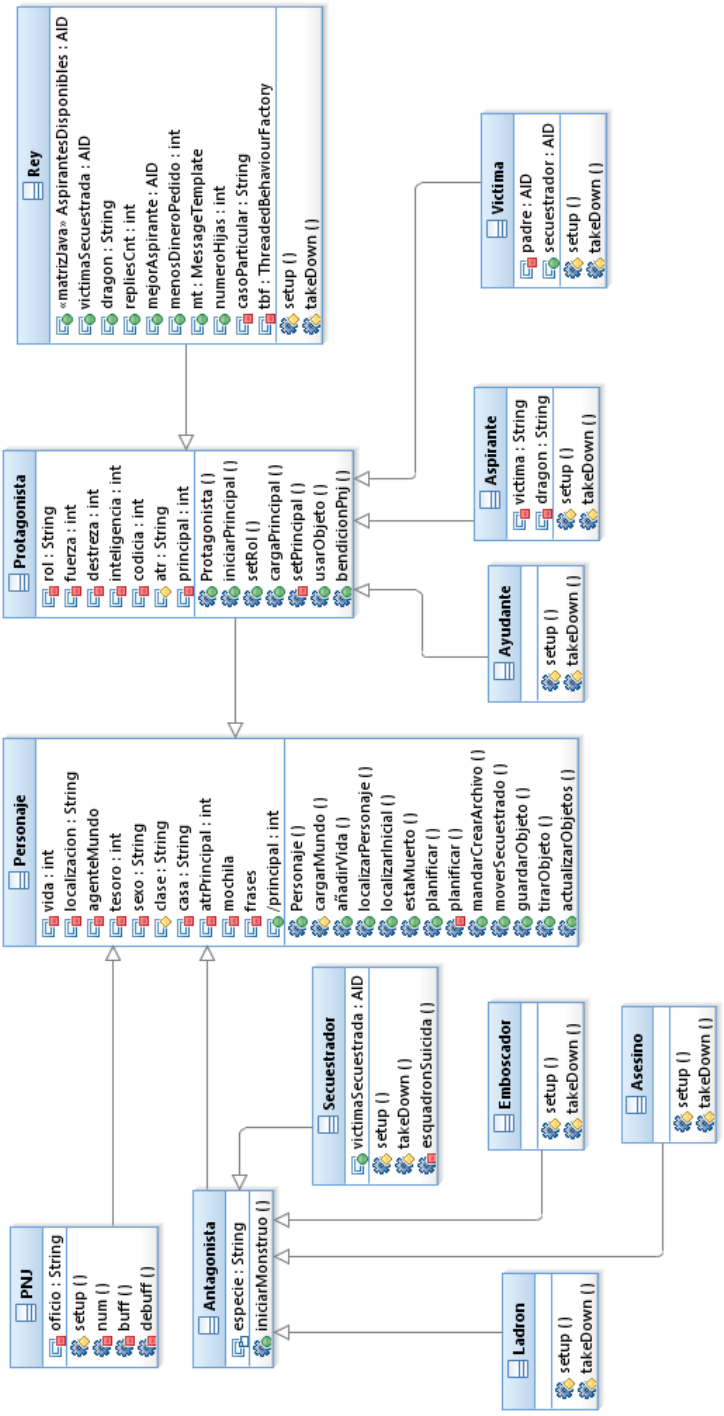


Figura 7.1: Diagrama de Clase de los Personajes

7.1.1. Acciones comunes a varios personajes

Sin embargo, los distintos personajes no comparten solo atributos y métodos, también tienen en común ciertas acciones que representan aquellas actividades que todos los personajes pueden hacer. Estas acciones no están incluidas dentro de la clase `Personaje`, pero sin embargo es a través de esta clase desde donde se invoca su funcionamiento, normalmente a través del método `planificar`. Ciertas de estas acciones son `Behaviours` de JADE, pero no todos pueden serlo debido a problemas de sincronización (para más información consultar el Capítulo 2.1.10) De igual manera, hay acciones específicas a cada tipo de personaje en concreto, pero esto se verá mas adelante en sus respectivas secciones localizadas en este capítulo.

Por ejemplo, está la acción `Batalla`. Esta acción expone como los personajes resuelven las disputas que tienen entre ellos. En el funcionamiento de una batalla, un personaje informa a su contrincante de que va a luchar a través de un mensaje ACL del tipo `Inform`. En ese momento ocurre la batalla. Para simularla se calcula la manera de golpear de cada personaje en específico. De esta manera, los protagonistas golpean restando la suma de su vida y su atributo principal a la vida de su contrincante, mientras que los antagonistas golpean restando directamente su vida. Aquel que consigue que la vida de su oponente sea menor que cero será el vencedor. En caso de que ambos lo consigan, ambos mueren.

También cuentan en su haber con el comportamiento `Mover`. Este comportamiento, permite a los distintos personajes moverse a través del mundo. Para ello, los personajes cuentan con su localización origen y su localización de destino y, a través un mensaje ACL del tipo `Request`, le piden al `Agente Mundo` poder moverse. El `Agente Mundo`, que conoce todo el mapa, comprueba que en efecto ambas localizaciones están conectadas y en caso afirmativo, informa al susodicho Agente a través de un mensaje ACL del tipo `Confirm`, de que puede cambiar su localización. En este mismo comportamiento el `Mundo` comprueba si hay un objeto en la nueva localización, en cuyo caso un `Agente Protagonista` podrá hacerse con él, y se comunica con el `Agente Emboscador` para su funcionamiento, lo cual se ve en detalle en la implementación de este mismo Agente más adelante en este capítulo.

7.2. Protagonistas

Para que la historia sea rica se necesita de no solo uno, sino varios personajes principales con los que disponer en el transcurso natural de la historia, pues cuantos más personajes haya, mejores historias se podrán crear. Lo natural en estos casos es que los protagonistas no sean meras copias unos de otros, así que se decide que dentro de los propios Protagonistas debe haber una nueva división de tipos.

7.2.1. Diseño de los Protagonistas

En la búsqueda de unos protagonistas para las historias se decide partir de aquellos ya planteados, como se ve en el capítulo 4, pero para que estos personajes sean originales y puedan cumplir papeles diversos en la historia es necesaria la inclusión del concepto de clase dentro de los protagonistas. Estas clases ayudan a parametrizar los pasos que seguirá cada clase de personajes dentro de la historia y así darles a cada clase ciertas funcionalidades específicas. De igual manera, también es importante remarcar que ciertos protagonistas son los únicos que pueden obtener y consumir recursos, tal y como se explica en el Capítulo 8 referente a los objetos, e interactuar con los PNJ's. En esta versión solo el Aspirante tendrá esta característica, convirtiéndose en un personaje consumidor. A continuación veremos los diferentes tipos de protagonistas.

7.2.1.1. La Princesa

La Princesa cumple un rol muy importante dentro de las historias. Ella siempre es raptada por un antagonista y a partir de ese momento se desencadena la acción de los distintos protagonistas repartidos por el mundo, y las diversas pruebas por las que deben pasar hasta que por fin es salvada. Sin embargo, la idea del personaje "Princesa" no acaba ahí, pues también se busca que sea un personaje capaz de actuar por iniciativa propia. Por tanto, en la búsqueda de la princesa fuerte e independiente, se decide que ésta también sea capaz de escapar por su propio pie. Así cuando los caballeros se demoren demasiado, la princesa se va impacientando mientras evalúa las posibilidades de éxito de una posible fuga, hasta que finalmente decide emprender la huida por su cuenta. Esto no quiere decir que la princesa se escapase siempre, simplemente a veces lo consigue y a veces no.

7.2.1.2. El Rey

El Rey cumple un papel más formal dentro de la historia. Su principal uso es el de tener un personaje intermedio para que los distintos personajes se puedan comunicar entre sí, pero no por ello es menos importante, ya que sin su ayuda la historia no puede transcurrir de manera fluida. La empresa del rey es la de contratar a los distintos caballeros del reino cuando alguna de sus hijas es secuestrada. Así, el Rey les pide ayuda a estos aspirantes a héroes con la esperanza de que alguno pueda rescatar a su preciada hija.

7.2.1.3. Los Caballeros

El propósito de los Caballeros en la historia es cumplir el papel de héroes. Para ello deben emprender su camino con el objetivo de rescatar a la

princesa de su malvado secuestrador, enfrentándose por el camino a las distintas pruebas propuestas por el resto de los antagonistas. En estas pruebas el caballero debe hacer uso de los diversos recursos que tiene a su alcance o incluso pedir ayuda a otros personajes, que forman parte de la historia como otros protagonistas, para poder superarlas.

7.2.1.4. El Mago

El Mago cumple el papel del personaje que ayuda en un momento crucial de la historia a los aspirantes a héroe de la misma. La forma de funcionar de este personaje se entiende cuando el caballero tiene un encuentro con un antagonista en específico, en este caso el ladrón. El caballero, que debe rescatar a la princesa cuanto antes y no tiene tiempo para enfrentarse al ladrón, pide apoyo a un mago, el cual acude en su ayuda bajo el precio de poder quedarse con el objeto que este antagonista ladrón mantiene vigilado.

Para encontrar un ejemplo más gráfico, imaginen una historia en la que el caballero ha de cruzar un lago en el que vive una gran sierpe que ha roto el puente. En ese momento un mago aparece en escena, congelando el lago junto a la serpiente, permitiendo al caballero cruzar el lugar tras lo cual el mago derrotará a la sierpe y encontrará entre los huevos que custodiaba un mágico amuleto de aspecto siniestro. Si sin embargo el mago no consiguiese derrotarla, la serpiente pelearía con el caballero, que si consigue vencerla puede proseguir su camino.

7.2.1.5. EL Héroe Caído ó Villano

El concepto de héroe caído, como ya se ha mencionado antes en el Capítulo 4.2.2, es el de un caballero con un corazón malvado que en última instancia decide no salvar a la princesa e intenta acabar con ella. El propósito de este personaje, aparte de cumplir la necesidad de los múltiples objetivos de un mismo agente tal y como se explico en el Capítulo 3, es el de no clasificar automáticamente a los personajes en "buenos" y "malos", pues siempre es un buen elemento narrativo las historias que cuentan con cambios inesperados en las actitudes de los personajes haciendo ver que no todo es como parece.

7.2.1.6. Protagonistas Adaptables

Como se explica en el Capítulo de Objetivos, una de la aspiraciones es conseguir que la aplicación sea altamente personalizable haciéndola lo menos restrictiva posible. Para ello se decide que los Protagonistas sean configurados mediante un archivo XML. Este archivo brinda la oportunidad de crear clases menos específicas, que se adapten a más entornos, en definitiva que sean más genéricas, para ello se cambia el nombre de las clases explicadas anteriormente pudiéndose ver el resultado en la Tabla 7.1.

Clase Protagonista	Papel en la Historia
Víctima (Princesa)	Será secuestrada en las historias
Allegado a la Víctima (Rey)	Pedirá que rescaten a la víctima
Aspirante a Héroe (Caballero)	Irá al rescate de la víctima
Ayudante del Héroe (Mago)	Ayudará al héroe en un momento crítico

Tabla 7.1: Tabla final de Clases de Protagonistas y su papel en la historia

Esta forma permite al usuario tener más flexibilidad a la hora de concretar los parámetros de su historia, de manera que puede elegir que uno de los héroes en su historia sea el caballero Don Quijote embarcado en el peligroso rescate de la princesa Dulcinea o que el samurai Kenshin deba rescatar a Mulán, la hija del emperador.

7.2.2. Implementación de los Protagonistas

Para poder desarrollar de manera eficiente los protagonistas se realizan numerosas tareas en la aplicación, ya que se decidió realizarla de cero, tal y como se explica en el Capítulo 4.3. Esto ha llevado a necesitar de distintas clases para poder cumplir los objetivos que se explican a lo largo del capítulo.

De igual manera, cada acción que aquí se comenta, como que la víctima pide ser rescatada, aparece en la historia, pero eso se ve con detalle en 9.

7.2.2.1. Carga y Configuración de los Protagonistas

Para poder tener unos protagonistas tan diversos se necesitaba de algo capaz de interpretar correctamente los datos de estos. Para ello se usa un SCD. Este SCD es capaz de cargar e interpretar los datos proporcionados a través del fichero "Protagonistas.XML", a partir del cual coge las características concernientes a cada Protagonista en específico y proporciona al `Agente Mundo` esos datos, recorriendo en el archivo todos los nodos con la palabra clave `<protagonista>`. El archivo de configuración tiene al final el aspecto del cuadro de Código 7.1.

Código 7.1: Formato XML de los protagonistas

```
<?xml version="1.0" encoding="UTF-8"?>
<Actores>
   <protagonista nombre="[nombrePersonaje]" clase="[nombreClase]"
      raza="[nombreRaza]" vida="[#]"
         fuerza="[#]" destreza="[#]" inteligencia="[#]"
            codicia="[#]"/>
</Actores>
```

7.2.2.2. La Clase Protagonistas

Para poder ejecutar correctamente a los protagonistas se necesita de una clase común para poder tratarlos a todos como el tipo de personajes que son. Esta clase interpreta correctamente los datos cargados desde el SCD de los protagonistas, para posteriormente enviarlos a cada clase en específico. Esta clase, que hereda a su vez de la clase `Personaje`, consultable en 7.1, tiene los atributos específicos de los protagonistas, y representa una gran ayuda a la hora de la implementación de estos. Una vez entendido bien el por que necesitamos de una interfaz común a todos los protagonistas, se va a proceder a explicar uno por uno como han sido realizados.

7.2.2.3. La Víctima

La ejecución de la clase empieza cargando los atributos y datos específicos del Agente. Tras esto, es necesario imponer a la víctima un pariente, consultando al DF por aquellos Agentes que registraron el servicio `Rey`, tras lo cual la víctima registra el servicio `Secuestrable` en el DF. Aparte de esto, posee tres comportamientos.

El comportamiento `mover-secuestrada` permite a la princesa moverse con su secuestrador, respondiendo al mensaje ACL `mover princesa`. Aunque a los ojos de la historia es el captor quien manda moverse a la víctima, el agente solo puede relocalizarse en el mapa a sí mismo, así que realmente se mueve por su propio pie, aunque, eso si, siguiendo las ordenes de su captor.

La víctima es raptada por un antagonista del tipo *Secuestrador*. Cuando la víctima es secuestrada, llega un momento en el que su captor la manda el mensaje ACL `Te Secuestro`, que le confirma que en efecto ha sido raptada. En ese momento la víctima lanza una petición de ayuda que es recogida por el pariente que se le ha asignado. Una vez esto ocurre la víctima intenta escaparse por sus propios medios. Para ello la víctima usa un comportamiento `TickerBehaviour` que envía cada X periodo de tiempo la petición `mujerIndependiente`, a través de un mensaje ACL del tipo `Request` a su secuestrador. Este mensaje contiene un numero aleatorio generado en cada periodo, que se encuentra en el rango de los valores de vida de su captor. Cuando el captor recibe el mensaje, evalúa si el numero recibido es mayor que su valor de vida actual ya que ante un secuestrador malherido, contaría con muchas más posibilidades de escapar. Si finalmente el número es mayor, la víctima consigue liberarse, informa al *Aspirante a Héroe* de que ya no necesita ser salvada y se planifica, tras lo que termina su ejecución.

Por último, tiene el comportamiento `rescatada`, que se ejecuta cuando es salvada, es decir, no se ejecuta cuando escapa por su propio pie. En este comportamiento, la víctima espera un mensaje ACL del tipo `Inform` con el ID `Rescatada`, que será lanzado por su salvador, indicando a la víctima el fin de su ejecución.

7.2.2.4. El Allegado a la Víctima

Al iniciarse el agente se carga un número aleatorio de víctimas a las que puede estar ligado, seguido de los datos específicos del Agente, con la particularidad de que también carga su salario específico, cualidad que usa el agente para contratar los servicios de los héroes dispuestos a salvar a la víctima, normalmente superior al del resto de personajes. Si el número de víctimas a las que puede estar ligado es cero, el agente se destruye.

Una vez comprobado que no existe ningún problema con los datos cargados, el Agente se localiza, tras lo que registra el servicio `Rey` y comienza su ejecución, incorporando dos comportamientos, `Tortura`, en el que se ve enfrentado al antagonista Asesino, y un `FSMBehaviour`.

En el comportamiento específico de `Tortura`, el agente evalúa si recibe el mensaje ACL `Agoniza` de un asesino, y en caso afirmativo empieza la ejecución de un `TickerBehaviour` que le va restando vida al agente progresivamente. En caso de que la vida llegue a cero, el comportamiento finaliza la ejecución del Agente.

A partir de aquí el funcionamiento del rey es ciertamente delicado. La ejecución del comportamiento `FSMBehaviour` se asemeja a la de una máquina de estados circular, donde cada comportamiento se ejecuta dando paso al siguiente. Para poder ejecutarse correctamente ha de registrar los diferentes estados, añadiendo a cada estado el comportamiento correspondiente e identificándolos por un nombre específico, y después añade las transiciones, que especifican el cambio entre estados. Los diferentes estados por los que puede pasar el agente son:

- **Atento**: En el estado atento, el Agente esta esperando a recibir un mensaje ACL del tipo `Request` proveniente de la víctima. Cuando este mensaje se produce, el Agente activa a un único antagonista del tipo `Asesino`, elegido de forma aleatoria, a través de un mensaje ACL del tipo `Inform` con el ID `Derrocar`.

- **Rescate**: En este estado el Agente busca en las paginas amarillas a aquellos Agentes con el servicio `Matadragones`, que serán los aspirantes a héroes, buscando al menos uno disponible.

- **Ayuda**: Aquí se registra un mensaje ACL del tipo `CFP`, que se envía a todos aquellos agentes encontrados en el estado anterior a través del `MessageTemplate Solicitar Servicio`.

- **Recibir Oferta**: El agente recibe la respuesta de cada `Aspirante a Héroe` a la petición del estado anterior a través de un mensaje ACL. En este mensaje, se incluye el precio por contratar los servicios de cada aspirante, tras lo que el `Allegado` elige aquel con la mejor oferta, el precio mas bajo. Si no posee suficiente dinero para pagar a un nuevo

aspirante, vuelve a buscar aspirantes nuevos comprobando nuevamente si cumplen los requisitos necesarios.

- **Aceptar Oferta**: Al aspirante seleccionado finalmente en el anterior estado, el *Allegado* le manda un mensaje ACL del tipo `AcceptProposal`, en el que le proporciona los datos de la víctima y del secuestrador, de manera que el aspirante a héroe sea capaz de planificarse para rescatar a la víctima. Tras esto el Agente espera una respuesta del aspirante en forma de mensaje ACL. Si este mensaje es del tipo `Inform` implica que el aspirante tuvo éxito en su empresa. Si no, significa que falló, y el `Allegado` volverá a buscar nuevos aspirantes.
- **Salvada**: En este estado el `Allegado` envía un mensaje ACL de tipo `Inform` a la víctima, indicándole que ha sido rescatada, y eliminándola de la lista de víctimas ligadas al `Allegado`. Si esta lista de víctimas queda vacía o el agente se queda sin dinero su ejecución terminará. Si no, volverá al estado inicial, Atento.

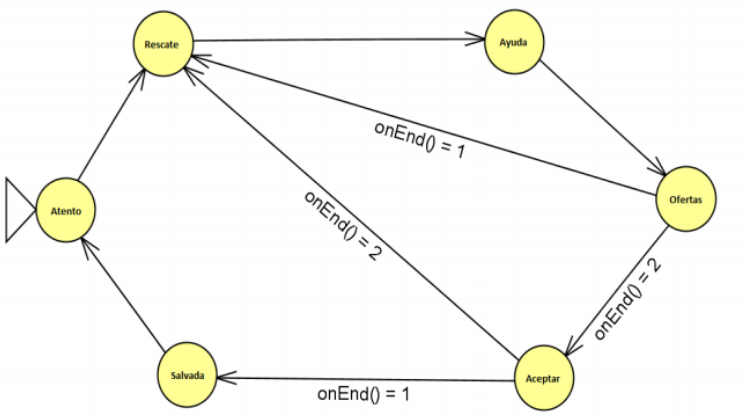


Figura 7.2: Diagrama de transición autómata del Allegado

Recuperado de Laclaustra et al. (2014).

7.2.2.5. El Aspirante a Héroe

Al iniciarse un nuevo aspirante, éste carga todos sus atributos, registra el servicio `Matadragones` y se localiza dentro del mapa. El agente registra dos comportamientos principales que son el comportamiento `OfrecerServicios()` y el comportamiento `AceptarOfertaRescate()`.

En el comportamiento `OfrecerServicio()`, el héroe espera un mensaje ACL del tipo `SolicitarServicio` que manda el *Allegado a la Víctima* cuando quiera que alguien salve a la víctima. A ese mensaje el héroe responde

enviando una propuesta, en forma de mensaje ACL del tipo `Propose`, indicando cuanto cuesta contratar sus servicios, que viene determinado por su codicia (véase el punto 7.5.1).

Si esta propuesta es aceptada accede a su siguiente comportamiento, `AceptarOfertaRescate`. En este momento le llega la información necesaria para cumplir el contrato, véase los datos del *Secuestrador* y de la *Víctima* que ha de salvar. Tras esto, informa con el mensaje ACL `ObjetivoSecuestro`, un mensaje de control del entorno que refleja que víctima será salvada por el héroe, y se planifica. Es dependiendo de su planificación cuando se decide si el aspirante a héroe acaba siendo un héroe o sin embargo, harto de la princesa, decide convertirse en villano.

Si decide convertirse en héroe, el Agente se enfrenta al `Secuestrador` en batalla y en caso de vencer escoltará a la `Víctima` hasta su hogar, tras lo que el `Aspirante` vuelve a su propia casa y se convierte finalmente en héroe.

Si decide convertirse en villano, el Agente se enfrenta al `Secuestrador` en batalla y en caso de vencer entablará batalla con la `Víctima` intentando acabar con ella. Si lo consigue se convierte finalmente en villano.

Independientemente de la elección, el Agente acaba su planificación y termina su ejecución.

Si en algún momento la víctima logra escapar por si misma, mandará a todos los aspirantes un mensaje ACL del tipo `Inform` activando su comportamiento `FinPlanificacion`, tras lo cual éstos pararán su ejecución, en caso de tenerla, y se replanificarán con el objetivo de volver a casa.

También es importante indicar que cada vez que el Agente se mueve por el mapa de una localización a otra, el entorno consulta a los antagonistas `Ladrones` y `Emboscadores` si les interesa interactuar en esta nueva localización con este Agente. Esto se ve con mas detalle en sus secciones respectivas.

7.2.2.6. El Ayudante del Héroe

El ayudante se inicia cargando todos sus atributos y registrando el servicio `Cazamagia`. Tras esto el Agente se localiza y añade un único comportamiento, `AyudaArcana()`.

El comportamiento `AyudaArcana()` es un comportamiento complejo, en el que participan varios Agentes y se comunican intercambiando mensajes. Este comportamiento realmente se activa cuando un `Antagonista Ladron` está en la misma localización que un `Aspirante a Héroe`, y es entonces cuando el `Agente Mundo` despierta al ayudante a través de un mensaje ACL del tipo `Inform`. En ese momento el ayudante se planificará, provocando que vaya al encuentro entre el ladrón y el aspirante a héroe, entablando batalla con el ladrón.

En caso de vencer al ladrón, el ayudante se quedará con su objeto y activará el comportamiento `ConvertirseEnSabio`, en el que tendrá que dejar

el objeto en un lugar seguro, es decir, su casa, tras lo que se convertirá en sabio, activando su comportamiento `FinPlanificaicon` en el que sencillamente acaba su ejecución.

7.3. Antagonistas

Para que la historia tenga sentido siempre debe haber algo que motive a los protagonistas a emprender sus acciones. Para cumplir este fin surge la figura del Antagonista, un personaje malvado con motivaciones que van en contra de aquellas que definen la manera de actuar de los protagonistas.

7.3.1. Diseño de los Antagonistas

Para diseñar los antagonistas se decide también que la mejor manera es clasificarlos según las distintas acciones que finalmente son capaces de realizar. Por tanto también se decide separarlos por clases capaces de diferenciar a los Antagonistas de una manera similar a lo propuesto con los Protagonistas. Las elecciones finales se explican en detalle a continuación.

7.3.1.1. El Dragón

El Dragón representa un personaje primordial en la historia. Al principio de la historia todo transcurre apaciblemente hasta que el dragón despierta, es decir que se inicia el Agente, y busca una víctima que pueda secuestrar. En ese momento el Dragón decide llevar a cabo el rapto dando pie a todos los acontecimientos que se suceden después, es decir, provocando que el rey avise a los caballeros y estos se muevan a través del mapa, que conforman el nudo de la historia.

7.3.1.2. Creando nuevos Antagonistas

Una de las propuestas más estudiadas es que nuevos héroes implican nuevos antagonistas. Sin embargo, estos nuevos antagonistas no pueden ser el mismo antagonista llamado de manera distinta (no sirve la idea de tener un grifo y un dragón para que ambos hagan lo mismo), y de igual manera no se ha querido introducir un nuevo tipo de antagonista con la capacidad de suplantar al Dragón como elemento primordial en la historia.

El primer enfoque explora el incluir antagonistas ligados a un lugar, es decir, antagonistas que vigilen un lugar específico del mapa y que castiguen a los aspirantes a héroes que deciden atravesar ese camino para completar su aventura. De esta manera, se crean localizaciones concretas y antagonistas específicos para ellas.

La primera localización elegida es el lago. En el lago puede vivir una Serpiente Marina, aunque en este caso es de agua dulce, que ataque a los aventureros que se atrevan a cruzarlo, con el objetivo de derrotarlos y que las aguas del lago descansen tranquilas siempre sin que nada las perturbe.

La segunda localización aprobada es el cruce. En este cruce de múltiples caminos, un avaricioso Trol monta guardia pidiendo a los confiados viajeros los tributos necesarios para poder continuar su camino. Los viajeros deben pagar el tributo o sino se enfrentarán a la ira del malvado Trol. Tras esto el Trol cambiará de lugar en el que emboscar, para poder preparar su siguiente emboscada y que nadie sea capaz de preverla.

Como tampoco se busca que los nuevos antagonistas se limiten a no dejar pasar a los aspirantes a héroe, se introduce un nuevo antagonista que no interactúa necesariamente con ellos, y así nace el concepto del Fantasma.

El Fantasma es el espíritu vengativo de un rey derrocado hace tiempo que no pudo saciar sus ansias de poder. Así su cometido será vagar por los castillos, buscando aquellos monarcas en los que la desesperación ha hecho mella, como aquellos con una hija secuestrada, alimentándose de su desdicha y conduciéndolos a la locura, lo que implica un descenso progresivo en sus puntos de vida.

Finalmente, se decide que el diseño de la serpiente no es lo suficientemente exclusivo, asi que se modifica para que busque un objeto, lo robe, y decida protegerlo en algún lugar atacando a los personajes que pasen por él.

7.3.1.3. Antagonistas genéricos

Como sucede a lo largo del proyecto, al tratar de darle un aire mas genérico a los antagonistas se decide que estos también deben ser configurados por el usuario a través de un archivo XML. Para ello la clasificación de los antagonistas dentro del sistema cambia, y se pasa a tener antagonistas secuestradores (dragón), emboscadores (trol), ladrones (serpiente) y asesinos(fantasma). Esta nueva clasificación permite que los antagonistas trabajen bajo las condiciones con las que antiguamente estaban diseñados pero que a su vez no tengan por que ser un antagonista impuesto por nosotros. De la misma manera, los antagonistas que actúan sobre una localización inicial, pueden reconfigurarla para que se adapte a los requerimientos del usuario.El esquema resultante se puede comprobar en la figura 7.2.

Antagonista	Objetivo
Secuestrador	Secuestrar Víctimas
Emboscador	Emboscar Héroes a cambio de dinero
Ladron	Robar y Proteger un Objeto
Asesino	Atormentar Allegados a Víctimas

Tabla 7.2: Tabla final de objetivos y tipos de Antagonistas

De esta manera el usuario a través del archivo XML, cuya construcción es explicada mas adelante, podrá dictaminar el nombre del antagonista, que clase de antagonista dentro de los tipos propuestos quiere que sea, que especie desea otorgarle y finalmente definir su sexo. Así, por poner algunos ejemplos, el usuario podrá elegir que la princesa Rapunzel sea secuestrada por el dragón Smaug o por la bruja Maléfica, o de la misma manera, que el dragón Smaug secuestre princesas en una simulación del sistema o bien embosque a los viajeros que atraviesan la Montaña Solitaria en otra simulación.

7.3.2. Implementación de los Antagonistas

Para implementar los Antagonistas se necesita tratar con agentes y sus propiedades. A través del `Agente Mundo`, los Agentes se comunican con otros Agentes y reciben la información que necesitan. Pero sin embargo para el correcto funcionamiento de los Antagonistas se necesitan muchas más cosas.

7.3.2.1. Carga y configuración de los antagonistas

Al crear el SCD de Antagonistas, capaz de cargar los datos correctamente del archivo "Antagonistas.XML", se sigue el patrón preestablecido en el Capítulo 5.4 con la particularidad de recorrer los nodos con la palabra clave `<antagonista>`, que contienen los atributos específicos de cada antagonista que se quiere introducir en la historia. La estructura del archivo XML se detalla en el Código 7.2.

Código 7.2: Formato XML de los antagonistas

```
<?xml version="1.0" encoding="UTF-8"?>
<Actores>
   <antagonista nombre="[nombreAntagonista]"
       clase="[Secuestrador|Ladron|Emboscador|Asesino]"
          especie="[nombreEspecie]" sexo="[M|F]"/>
</Actores>
```

7.3.2.2. La Clase Antagonistas

La clase `Antagonistas` es aquella capaz de interpretar correctamente un agente mandado desde el SCD de antagonista. Esta clase, que a su vez hereda de la clase `Personaje`, consultable en la Sección 7.1, es donde se consigue cargar todos los atributos relativos al antagonista, como su especie, su localización y su vida, y finalmente le dejaba el trabajo a la clase en cuestión que se encargaba de ese tipo de antagonistas en concreto. A continuación se entrará en detalle a explicar la implementación de cada antagonista en concreto, especificando que comportamientos y acciones es capaz de desarrollar como agente.

7.3.2.3. El Antagonista Secuestrador

Al iniciarse el `Agente Secuestrador`, carga todos sus atributos, localizandose en el mundo y añadiendo el comportamiento `Secuestro`.

Según se inicia el comportamiento `Secuestro`, el `Agente Secuestrador` solicita información al `Agente Mundo`. En este caso el `Mundo` tiene que indicar cuales son aquellos Agentes que proporcionan el servicio `Secuestrables` y, en caso de que no haya ninguno, el Agente se queda a la espera de que aparezca uno que cumpla los requisitos solicitados.

En caso de encontrar al menos uno de los agentes con el servicio secuestrable, escoge uno de manera aleatoria eligiendo así su objetivo, indicando al entorno mediante el mensaje ACL `ObjetivoSecuestro` quien es la víctima elegida. Una vez ha escogido el agente a secuestrar se planifica, y al finalizar incorpora a su funcionamiento los comportamientos de `FalloSecuestro`, que entra en acción cuando falla en el rapto, y `FinPlanificación`, que se ejecuta cuando el secuestro se realiza correctamente.

Cuando se ha producido un error durante el secuestro, el agente informa al `Agente Mundo` del mismo a través de un mensaje ACL del tipo `Failure` que indica al entorno el fallo, tras lo cual el Agente detiene su ejecución y vuelve al punto inicial, es decir que esperará a que aparezca algún Agente secuestrable.

Sin embargo, cuando el secuestro se realiza con éxito, el `Agente Secuestrador` se dispone a finalizar correctamente su ejecución. En ese momento informa al `Agente Mundo` del rapto y a través de un mensaje ACL del tipo `Inform`, notifica al `Agente Victima` que ha secuestrado que en efecto ha sido raptado, provocando que la víctima avise al allegado al que esta ligado. Por último, vuelve a su guarida, arrastrando consigo al `Agente Victima`, donde esperará a que alguien venga a por él y finalmente incorporará el comportamiento `VigilaTuEspalda`.

Este comportamiento permite al `Agente Victima` secuestrado escaparse de su raptor. Para ello, la víctima genera un número aleatorio entre los posibles valores de vida de su captor. Cuando el número generado es mayor que la vida actual de éste, escapará, lo cual implica que es mucho mas fácil escapar de un captor malherido que de uno que aún conserve todas sus fuerzas.

7.3.2.4. El Antagonista Emboscador

El `Agente Emboscador` carga todos sus datos y registra el servicio `Emboscador` en el DF que le permitirá actuar como tal, tras lo cual añade los comportamientos de `Emboscar` y `Acecho`, y finalmente se planifica.

El comportamiento de `Emboscar` es bastante complejo. Lo primero que se debe contemplar es que el `Agente Emboscador` ha sido colocado en el mundo, consultar el Capítulo 6.1.2 para conocer como, en la localización donde realiza sus emboscadas y que solo embosca a personajes de una clase en específico, en este caso los *Aspirantes a Héroe*. Tras esta aclaración se explica como funciona el comportamiento de Emboscar.

Emboscar realmente es una negociación entre el `Agente Emboscador` y el `Agente Mundo`. Cada vez que un personaje se mueve de una localización a otra, el `Agente Mundo` le manda un mensaje ACL al `Agente Emboscador` del tipo `CFP` con la información relativa al tipo de Agente que esta cruzando. Si este tipo de Agente no interesa al emboscador, este respondera con un mensaje ACL del tipo `Refuse`. En caso contrario, responderá con un mensaje ACL del tipo `Propose` en el que enviará su localización. Si ambas localizaciones coinciden, el `Agente Mundo` aceptará la propuesta proporcionandole al `Agente Emboscador` la información específica del Agente que se ha desplazado a esa localización. En caso de haber varios Agentes emboscadores, el mundo elegirá uno aleatoriamente y negociará con ellos por orden.

Una vez que la propuesta del `Agente Emboscador` ha sido aceptada, da comienzo el comportamiento `Acechar`. En el comportamiento `Acechar`, el `Emboscador` tiene los datos del Agente al que esta emboscando, proporcionados por el `Agente Mundo`. De esta manera genera un número aleatorio con el dinero que le pide al otro Agente, a través de un mensaje ACL del tipo `Request`. Si el otro Agente tiene el dinero suficiente paga y pasa sin problemas, sino el `Emboscador` lo mata. Independientemente del resultado, el `Emboscador` se planifica, resituándose dentro del mapa en una nueva localización en la que emboscar.

7.3.2.5. El Antagonista Ladrón

Al iniciarse el antagonista `Ladron` carga todos sus atributos, tras lo que se localiza en el mundo y añade el comportamiento `Acecho`.

El comportamiento `Acecho` es un comportamiento sencillo que provoca que el antagonista se planifique. Así que este, a través de su planificación, se mueve a la localización donde está el objeto que quiere robar, el cual es elegido de manera aleatoria, momento en el cual se pone a defender susodicho objeto, de manera similar a como el `Secuestrador` protege al `Agente Víctima`, atacando a cualquier aspirante a héroe que pase por la localización en la que se encuentra. En ese momento, el `Ladron` activa al ayudante del héroe a través del mensaje ACL del tipo `Inform HoraMagica`. Tras esto, el ayudante acude y es realmente quien tiene la batalla contra él.

7.3.2.6. El Antagonista Asesino

Cuando se inicia este antagonista, carga todos sus datos, se localiza y registra el tipo `Maligno`. Tras esto añade el comportamiento `Maldad`.

El comportamiento `Maldad` es un comportamiento cíclico(`CyclicBehaviour`), es decir, que ocurre cada cierto tiempo. En él, el antagonista `Asesino` elige a aquel `Allegado a la Víctima` cuya `Víctima` haya sido capturada, a través de un mensaje ACL del tipo `Inform`, y por tanto es derrocable. Tras esto se planifica, ejecutando la acción `Tortura`, indicando al allegado a través del mensaje ACL del tipo `Inform Agoniza` que su vida va disminuyendo poco a poco, concretamente cada 30 segundos.

7.4. PNJ

> *Los PNJ's son personajes del juego que son controlados por el sistema y que son capaces de interactuar con el avatar del jugador intercambiar artículos e información.* Gómez-Gauchía y Peinado (2006).

En el sistema se ha querido adaptar estos personajes característicos de los juegos a la narrativa para conseguir mayor diversidad de historias generadas.

7.4.1. Diseño de los PNJ's

En este punto el número de historias es altamente variado. En busca de un nuevo aire que dé un toque distinto a las historias y aumente en cantidad el número posible de generaciones de estas, se plantea la idea de unos personajes simples que con una acción sencilla cambien ligeramente el curso de la historia o que sencillamente aparezcan en ella dándole un toque divertido a ésta.

Para ello se propone crear numerosos personajes, que tengan como peculiaridad el interactuar con los protagonistas cuando éstos pasan por las localizaciones en las que están situados. En ese momento se produce un pequeño diálogo entre ambos personajes, otorgándo el PNJ un objeto, 8.1.3, al personaje principal. Este aspecto dentro de la historia es un modificador de atributos para los protagonistas, que bien puede ser afectar de manera positiva por que se ha encontrado con un PNJ amistoso o de manera negativa si éste es malévolo.

Para terminar de definir los PNJ's se ha pensado que lo que más encaja dentro de la propuesta de diseño es que éstos tengan un oficio. De esta manera, al crear los PNJ's se propone que para definir el objeto con el que interactuan lo suyo es clasificarlos por su oficio, de manera que se decide que los PNJ's sean herreros, enfermeras o sastres, que viven en lugares relacionados con su profesión, tal y como se comento en el Capítulo Mapa en el

apartado 6.1.2(véase la herrería, el hospital o la sastrería) y proporcionen este modificador con algo relacionado a su campo, por ejemplo el herrero puede forjar una espada mejor o reestablecer las abolladuras de la armadura del personaje, lo que se acaba representando como una mejora en un atributo del propio personaje, como puede ser la fuerza con la espada o la vida con la armadura. De la misma manera, y para que entendamos el concepto de PNJ malévolo, el personaje podría cruzarse con un chamán, que le da una pócima de setas "bendecidas por los ancestros", donde en unas simulaciones aumentarán los atributos del personaje y en otras los disminuirán.

Sin embargo, se observa que este diseño limita a los PNJ's a unos personajes muy específicos con un guión demasiado particular, características que se han querido evitar dentro de la aplicación. De esta forma, se decide que los PNJ's también puedan ser configurables por el usuario a través de ficheros XML que permiten al usuario personalizar su propio ámbito.

7.4.2. Implementación de los PNJ's

La implementación de los PNJ implica numerosas adiciones de clases nuevas al proyecto. Cada una cumple con una funcionalidad específica, y son la mejor solución para cumplir satisfactoriamente el papel de los PNJ's dentro de la aplicación.

7.4.2.1. PNJ's configurables por el usuario

Para permitir la configuración de los PNJ's a través de los XML se necesita, como de costumbre, de un SCD capaz de cargar los datos del archivo y almacenarlos correctamente dentro de la aplicación.

Este SCD sigue el diseño normal del resto 5.4, con la particularidad de que carga específicamente el fichero "PNJs.XML", donde recorre sus nodos buscando la palabra clave `<pnj>`. Una vez encontrada, recorre el nodo que representa al propio PNJ almacenando la información referente a su nombre, su sexo, su oficio y su localización, usando esta información para crear un nuevo agente con estas características. Como con todos los archivos XML, esta configuración exige que el fichero esté creado de una manera concreta, consultable en el fragmento de Código 7.3, por lo que se incluye en el fichero "PNJs.XML" una guía sobre como construir estos agentes, lo mismo que viene rellenado con una información por defecto que también puede usarse como ejemplo.

Código 7.3: Formato XML de los PNJ's

```
<?xml version="1.0" encoding="UTF-8"?>
<Actores>
   <pnj nombre="[nombrePNJ]" sexo="[M|H]" oficio="[nombreOficio]"
      localizacion="[nombreLocalización]"/>
</Actores>
```

7.4.2.2. La clase PNJ

Si bien el Agente se inicia desde el SCD, para poder cargarse de manera correcta necesita de una clase capaz de usar correctamente las características específicas de este tipo de Agentes. De esta manera se crea la clase `PNJ`, que hereda de la clase `Personajes` 5.5, donde se incluyen los métodos específicos que permiten el buen funcionamiento de los PNJ's.

De la misma manera en esta clase se incluyen los métodos que determinarán la manera de actuar de los PNJ's y es capaz de manejar como se desarrollará su ejecución.

Al empezar su ejecución un PNJ carga sus datos, tras lo que se localiza y se mantiene a la espera en esta localización. Cuando un personaje consumidor, que son aquellos personajes capaces de coger objetos e interactuar con los PNJ's, pasa por esta localización, el `Agente Mundo` avisa al PNJ a través de un mensaje ACL del tipo `Request`.

Cuando al PNJ le llega este mensaje, decide de manera aleatoria si el personaje recibirá una bendición o una maldición. En cualquier caso se generarán de manera aleatoria unos valores que modificarán los atributos del personaje, para mas información consultar el Capítulo 7.5.1, sumando estos nuevos valores a los que ya poseía el personaje en caso de ser una bendición y restándolos en el caso contrario.

7.5. Configuraciones de los Personajes

En una historia no debería haber dos héroes iguales, donde uno fracasa otro debe tener la oportunidad de vencer y viceversa. Tampoco los mismos héroes nacidos en distintos lugares deberían ser iguales, parecidos sí, pero no iguales, e incluso dentro de la misma zona, puede haber héroes que destacan más en la batalla que otros.

El objetivo de que los personajes se distingan entre sus iguales se ha realizado a través del concepto de atributos y razas.

7.5.1. Diseño de los Atributos y las Razas

Con animo de hacer a los personajes aun más exclusivos, se decide introducir atributos que representan su forma física y mental. Estos cambios realmente solo afectan a los protagonistas, pues los antagonistas tienen sus propios atributos y los PNJ's no se ven envueltos en conflictos Los atributos implementados son:

- Vida: Atributo referente a la capacidad de aguante de un personaje.
- Fuerza/Destreza/Inteligencia: Atributos referentes a la capacidad de combate del personaje.

- Codicia: Atributo que define el coste de contratar a dicho personaje.

Estos atributos representan modificadores positivos a la hora de enfrentarse a un problema. Por ejemplo, la fuerza será un factor diferencial que dictamina las posibilidades de victoria a la hora de enfrentar a un antagonista en batalla, mientras que la codicia representa el precio que un héroe pide a cambio de contratar sus servicios.

De la misma forma, se propone el concepto de raza, que es una característica más que tienen los personajes protagonistas. Esta nueva característica modifica según ciertos multiplicadores los atributos que un personaje posee, haciendo así que dos héroes de razas distintas tengan muy probablemente atributos distintos.

La manera en que los atributos y las razas se relacionan viene dado porque los personajes de ciertas razas tienen atributos más acentuados que otros. Esto implica que, por ejemplo, los personajes de la raza X resulten especialmente fuertes y los de la raza Y especialmente inteligentes, lo cual no quiere decir sin embargo que no exista un personaje de la raza Y especialmente fuerte que supere a otro de la raza X que no tenga la misma fortuna.

De igual manera, la clase también afecta a estos atributos, ya que un caballero en principio tendrá más opciones de ser fuerte que una princesa, ya que para poder llevar esa flamante armadura también hay que echar sus horas en el gimnasio, lo cual nuevamente no implica que no pueda haber una princesa especialmente fuerte, sino que ese caso seria una extraña variedad dentro de las múltiples historias que se generan.

7.5.2. Implementación de los Atributos y las Razas

Para explicar la implementación de los atributos es necesario comprender bien la clase `Personaje`. Estos atributos se cargan a la hora de crear el personaje y vienen dictaminados por los diferentes archivos XML en los protagonistas, así como por su modificador racial, pues solo los protagonistas tienen esta característica. Cada tipo de personaje tiene un atributo principal, que representa su manera de afrontar la batalla, por ejemplo, un aspirante a héroe emplea la fuerza para luchar mientras que un ayudante usa la inteligencia destreza. La Tabla final queda de la siguiente forma:

Clase de Personaje	Atributo Principal
Víctima	Aleatorio
Allegado a la Víctima	Codicia
Aspirante a Héroe	Fuerza
Ayudante del Héroe	Inteligencia ó Destreza (el mayor)
Antagonistas	Vida

Tabla 7.3: Tabla de Atributos Principales

Para el caso de los Antagonistas sin embargo, sólo interesa el atributo Vida, que es el atributo que se necesita conocer para calcular que un personaje pueda o no derrotarlos. Los PNJ sin embargo no cuentan con dichos atributos, ya que su aportación en la historia no les permite entrar en conflictos con otros personajes.

Las razas y sus características vienen dictaminadas de un archivo de configuración XML donde se especifica el nombre de la raza y los distintos multiplicadores a atributos que esta representa.

7.5.2.1. Razas configurables

Para realizar correctamente la configuración de las Razas a través de un archivo XML se necesita, nuevamente, un SCD capaz de cargar e interpretar los datos.

Este SCD sigue el patrón normal de todos los SCD 5.4, cargando el fichero específico "Razas.XML", y en su recorrido de nodos buscaba la palabra clave `<raza>` a partir de la cual carga las características específicas de esta, que son enviadas a un objeto `Raza` e introducido a una lista de las distintas razas.

La configuración específica que debe tener este fichero para que la aplicación funcione correctamente debe seguirse a rajatabla, por lo tanto junto al archivo de configuración se incluye un prototipo propio de configuración de razas que podrá usarse como ejemplo aparte de una pequeña guía de como ha de construirse el archivo, consultable en el cuadro de Código 7.4.

Código 7.4: Formato XML de las razas y atributos

```
<?xml version="1.0" encoding="UTF-8"?>
<Actores>
<raza nombre="[nombreRaza]" vida="[#]" fuerza="[#]" destreza="[#]"
inteligencia="[#]"  codicia="[#]"/>
</Actores>
```

7.5.2.2. La Clase Raza y como interactúa en la carga del personaje

La clase `Raza` permite interactuar con los atributos específicos de cada raza a la vez que guarda todos aquellos relativos a una.

Cuando un protagonista se crea, accede a la interfaz `Vocabulario`, donde existe un *HashMap* de razas en el cual cada nombre de raza va ligado a su objeto `Raza` correspondiente. Así el protagonista accede al nombre de la raza a la que pertenece, atributo que le viene dado por su configuración XML, y carga ese objeto `Raza`. Finalmente, a la hora de crear sus propios atributos, accede a aquellos atributos que le vienen dados por su configuración XML y los multiplica por los modificadores que le otorga su raza, dando esta operación las cifras finales de los atributos que posee.

7.6. Narrativa de los personajes

Si se observa dentro del campo de la generación de narrativa computacional la mayoría de sistemas están ligados a un único tipo de historias, como se puede observar en el punto 2.3. Continuando el afán de crear un sistema altamente configurable y que no este enfocando en un único ámbito se ha realizado el siguiente diseño.

7.6.1. Diseño de la narrativa

Para poder modificar la narrativa del generador de una forma fácil con la que esta pueda ser modificada rápidamente se ha realizado un diseño en donde la narrativa no este acoplada al código del generador. Para poder llevar a cabo este objetivo se almacena la información de forma externa y se carga al ejecutarse el generador. Con este diseño se consigue desligar el apartado narrativo del apartado técnico.

Además, debido a que el generador esta enfocado las historias entre los personajes, van a ser estos mismos los encargados de almacenar toda la información lingüística que conocen

7.6.2. Implementación de la narrativa

Para poder llevar a cabo el diseño pensado se han creado dos clases diferentes, una clase `Frases` y otra `Diccionario`.

La clase `Frases` es la unidad básica que se ha creado para que puedan los `Personajes` trabajar con la narrativa. El atributo principal de esta clase es un *HashMap* en el cual la clave es un *String* que hace referencia a las acciones de los personajes, que se vincula a un *ArrayList* de *String* y es aquí donde se almacenan todas las posibles frases que puede utilizar un personaje al realizar una acción en concreto. Al almacenarse las frases en una lista permite que se utilicen diferentes frases para una misma acción. Al invocarse el método `getFraseAccion`, este devuelve una frase aleatoria, por lo que a cuantas más frases pueda utilizar mayor variedad tendrá la narración de la historia dificultando así que un mismo personaje tenga un guión preestablecido y sea prácticamente imposible que se refleje la misma historia cuando un personaje realiza una acción.

La clase `Diccionario` es la estructura que utiliza el `Mundo` para que se encargue de tener almacenadas todas las frases que se utilizan a la hora de generar la historia. Estas frases están vinculadas a los hijos de la clase `Personaje` en función de un atributo específico(Ver Tabla 7.4).

Personaje	Atributo
Protagonista	Clase
Antagonista	Especie
PNJ	Oficio

Tabla 7.4: Tabla de la narrativa en función del personaje y atributo

También se necesita vincular las frases a una acción determinada. Esto es debido a que las frases se invocan dentro de las clases almacenadas en la carpeta `acciones`, donde se encuentran todas las acciones y *behaviours* los personajes pueden realizar. Es dentro de estas clases donde se invoca al método `hablar`, que heredan todos los Agentes, de la clase padre `Personaje`.

7.6.2.1. Carga en el sistema

Como se ha mencionado previamente la información se quiere guardar de forma externa al código y es el `Mundo` el encargado de almacenar esta información, por ello se ha creado, una vez más, un SCD. Una vez el `Mundo` se crea invoca a este SCD, el cual se encarga de cargar la información narrativa del sistema de un archivo XML, "Frases.XML", con la estructura que se ve en el Código 7.5. Dentro del XML hay una sencilla guía explicativa de la estructura de este.

Código 7.5: Formato XML de la narrativa de los personajes

```
<?xml version="1.0" encoding="UTF-8"?>
<Frases>
   <personaje tipo="[NombreAtributo]">
      <frase accion="[NombreAccion]">
         <f>[ frase que se quiere utilizar ]</f>
      </frase>
   </personaje>
</Frases>
```

Actualmente la información se almacena en el XML función de un atributo tipo ligado a los personajes, ver Tabla 7.4, para trabajar en un entorno controlado, pero podría hacerse un archivo XML más extenso para que la lingüística de cada personaje fuese única, vinculando las frases con atributos más específicos y/o con mayor cantidad de atributos.

Por tanto, el SCD busca en primer lugar la etiqueta `<personaje>` con el atributo `tipo` para empezar a almacenar en el `Diccionario` del `Mundo` por este atributo. Acto seguido se busca otra etiqueta, `<frase>`, dentro del bloque anterior por el atributo `accion` que es la clave para el *HashMap* vicunlado a la clave `tipo` del `HashMap` de `Diccionario`. Por último, dentro de estos dos bloque se busca una etiqueta `<f>` que tiene dentro la frase que se desea guardar en el generador y que son las que finalmente se ven reflejadas en la generación de la historia.

7.6.3. El guión de la narrativa

Es importante no olvidar lo necesario para que exista un transcurso de una historia y es que los personajes tengan que intentar cumplir unos objetivos. Anteriormente en este Capítulo se ha hablado sobre que objetivos específicamente intentaran cumplir tanto los Protagonistas como los Antagonistas, ya que los PNJs no se crean con la finalidad de cumplir objetivos que sean un hilo en la historia. Pero el objetivo que tendrá que cumplir cada personaje no está en su conocimiento hasta poner en marcha el sistema. Al arrancar el sistema cada personaje que se inicia llama al SCD de objetivos y es en ese momento que conoce que objetivo debe cumplir cuando ya puede pasar a planificarse.

El archivo que utiliza el SCD es un XML llamado "Objetivos.XML" y se puede ver su estructura en el cuadro de Código 7.6. El SCD busca la etiqueta `<personaje>` con el atributo `tipo`, este atributo hace referencia al tipo de personaje que es, por ejemplo Allegado, Secuestrador, etcétera. Y acto seguido busca la etiqueta `<objetivo>`. Dentro de esta etiqueta se debe encontrar una cadena de caracteres que hace referencia a una acción la cual el planificador tiene que ser capaz de entender, es decir que tiene que estar contemplada en el archivo "domain.PDDL". Un ejemplo es que para un Aspirante se encuentre el objetivo `(and (esHeroe Aspirante) (salvada Victima))`.

Código 7.6: Formato XML de los objetivos de los personajes

```
<?xml version="1.0" encoding="UTF-8"?>
<Objetivos>
   <personaje tipo="[TipoDePersonaje]" >
      <objetivo>([Objetivo])</objetivo>
   </personaje>
</Objetivos>
```

7.6.4. Ventajas del diseño

La elección de este diseño ha sido para evitar el acoplamiento entre la parte técnica de la aplicación y la parte narrativa. El hacer esto da al sistema muchas variantes, ya que cambiando un único archivo de configuración se puede cambiar la forma narrativa de la historia. Así en función del escritor se tendrá una narrativa generada de forma personal. También cabe destacar que este diseño permite introducir un archivo en diferentes idiomas, haciendo útil el sistema a nivel mundial.

También es un factor importante el que los personajes no conozcan su objetivo hasta el momento de iniciarse, de esta forma se evita el saber de antemano que misión tiene que realizar cada uno de los personajes creados o como es en el caso del Aspirante si será un héroe o un villano.

Capítulo 8

Diseño e Implementación de los objetos

Un Anillo para gobernarlos a todos. Un Anillo para encontrarlos, un Anillo para atraerlos a todos y atarlos en las tinieblas.

Tolkien, 1993

Objetos... cuan de importantes son en muchas historias. Quien no recuerda a Excálibur, la legendaria espada del rey Arturo, o al Anillo Único de Sauron en los libros de Tolkien. Los objetos pueden ser el objetivo de una historia o dar ese giro inesperado que cambia el transcurso de la historia. En este proyecto se ha querido introducir objetos para crear historias mucho más ricas.

8.1. Diseño de los objetos

En este punto del proyecto las historias que se están generando tienen peso y dinamismo, por todo lo que se ha explicado en puntos anteriores, pero con la idea de seguir consiguiendo más enriquecimiento en las historias se ha tenido en cuenta la inclusión de objetos. Los objetos se introducen con el fin de conseguir historias más complejas y aparte también reflejar mejor los conflictos entre personajes. Por ejemplo, si en un enfrentamiento entre dos personajes uno anteriormente ha conseguido una espada, este tendrá más oportunidades de salir vencedor al tener un arma que le dará ventaja en una batalla.

8.1.1. Objetos consumibles

El primer planteamiento que se tuvo en mente de los objetos fue el de que fuesen objetos para usar en el momento en el que se encuentran obligatoriamente. Estos objetos encontrados se consumirán en el acto y desaparecerán del mapa, alterando las estadísticas del personaje en el momento de su consumición. Ver Capítulo 7.5.1.

Este planteamiento sirvió como primer paso para introducir los objetos en el proyecto, aunque se pensó que los objetos diseñados de esta manera quedaban en la historia de forma efímera.

8.1.2. Objetos clave

Con este pensamiento se introdujo un segundo tipo de objetos que serían clave durante el transcurso de la historia para poder lograr los objetivos de algunos personajes. Por ejemplo para que el caballero pueda entrar en la montaña para rescatar a la princesa necesitará un objeto que le servirá como llave para poder entrar. Gracias al diseño de este nuevo tipo de objetos, habrá objetos que tendrán un papel relevante en la historia, al igual que los objetos que se han mencionado anteriormente en la introducción de este capitulo.

8.1.3. Objetos creados por los PNJ

En este punto ya se tienen diseñados dos tipos de objetos que cubren las necesidades actuales del proyecto y se observa que tanto los personajes de tipo protagonista como los de tipo antagonista interactúan con los objetos tal y como están diseñados hasta este punto, pero no los PNJ. Debido a esto y ya que los PNJ están configurados como entidades que tienen un oficio, no es descabellado hacer que estos fabriquen objetos, como que un alquimista cree una poción, un tabernero sirva una cerveza o un armero fabrique un arma. Estos objetos fabricados por los PNJ están pensados para tener un funcionamiento idéntico a los objetos consumibles,l es por esto que los objetos que fabriquen los PNJ solo se aparecerán en la historia a través de dicho PNJ, es decir no se podrá encontrar mientras se explora por el mapa, y de esta forma diferenciar ambos tipos de objetos.

8.2. Implementación de los objetos

Como el resto de componentes de la aplicación, los objetos han sido implementados de forma altamente configurable y extensible. Se va a explicar detalladamente los pasos seguidos para conseguirlo.

8.2.1. Sistema de clases

Para poder trabajar con los objetos se ha creado una clase padre, `Objeto`, de la que extienden dos hijas, `Clave` y `Consumible`. Ambas clases hijas tienen en común un atributo identificativo y otro descriptivo además de los correspondientes métodos *getters* y *setters*. Luego por un lado los `Objetos Clave` tienen un atributo que permite saber en que localización se encuentran dentro del mapa, mientras que por otro lado los `Objetos Consumible` almacenan la información referente a los atributos que afecta y de que manera afecta.

Una vez se ha creado como serán los objetos hay que situarlos en el mapa. Para esto se ha creado una nueva clase llamada `Almacen`, en donde se almacenan los `Objeto`, en una lista, en función de si son `Clave` o `Consumible`. Esta nueva clase permite tener un atributo a `Localizacion` para saber que objetos contiene en cada momento. Cuando los objetos ya están cargados por el sistema se introducen los `Consumible` en cada localización de la escenografía aleatoriamente. Con esto se consigue que en cada simulación no se encuentre un objeto concreto siempre en la misma localización. Para observar de forma concisa toda esta información referente a la arquitectura del sistema de clases de los objetos se ha incluido la Figura 8.1.

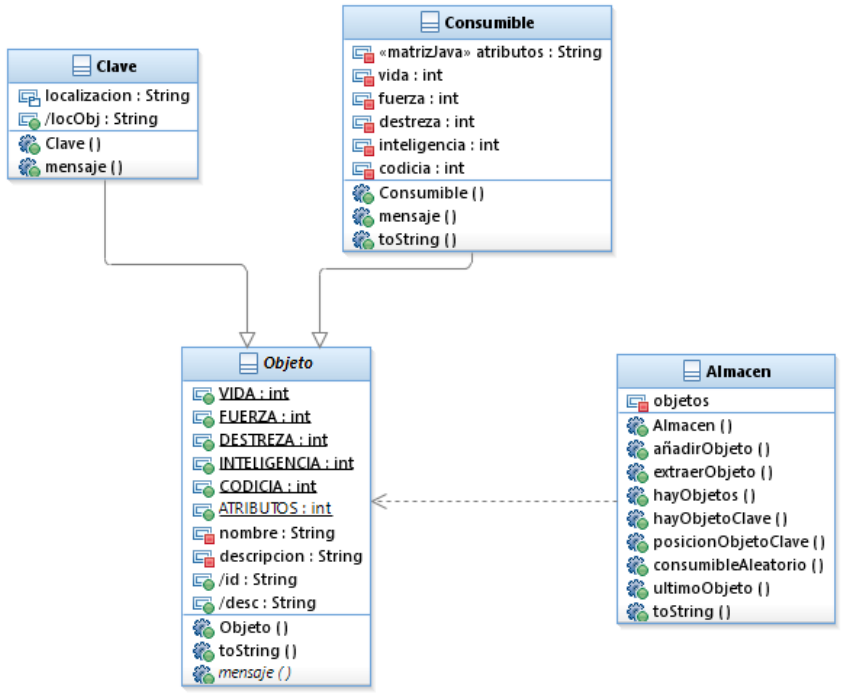


Figura 8.1: Diagrama de Clase de los Objetos

Ahora que ya hay implementados dos de los tipos de objetos diseñados, queda explicar como funcionan los objetos creados por los PNJ. Para estos objetos no se ha creado una clase diferente, que en la fase actual del proyecto no se ha querido que estos objetos se almacenasen, ya sea por los personajes o en las localizaciones. Por ello los objetos de los PNJ no se crearán realmente si no que cuando hay un encuentro entre ciertos personajes y un PNJ, este le aplicará directamente una modificación en los atributos, de la misma forma que un objeto consumible. De esta forma queda abierta una vía para ampliar este campo en futuras ampliaciones, haciendo por ejemplo el PNJ a la vez que genera objetos los vaya distribuyendo por diferentes localizaciones del mapa.

Con todas estas clases nuevas en un limbo se necesita crear una conexión para que puedan ser usados por los personajes y es a través del `Estado` que los objetos se integran en el sistema. Esto se debe a que los objetos clave también están involucrados en la planificación, ya que si un personaje necesita un objeto para cumplir un objetivo el planificador tiene que conocer dicho objeto. Además ya que solo se desea que ciertos personajes puedan consumir objetos es el `Estado` quien controla este aspecto mediante un listado de personajes consumidores, de esta manera cuando un personaje se encuentra en la misma localización que un objeto consumible y/o un PNJ para poder recibir una modificación en las estadísticas, el `Estado` mira el listado de consumidores y en caso de estar el personaje en el listado el `Estado` informa al personaje de que tiene un objeto o PNJ con el que puede interactuar. También el `Estado` es quien recibe y almacena toda la información que se encuentra guardada externamente.

8.2.2. Carga en el sistema

Una vez más se ha implementando un SCD. Al igual que el resto de SCD, este también extrae de un archivo XML, `Objetos.XML`, los datos que usa el sistema para poder trabajar con objetos.

El SCD empieza buscando en primer lugar la etiqueta `<tipo>` con el atributo `tipo` para guardar en el `Almacen` del `Estado` los objetos en función de este atributo. El siguiente paso es buscar la etiqueta `<objeto>` dentro del bloque anterior y en función del valor del atributo `tipo` busca unos atributos u otros. Si el valor es *clave* carga el valor del atributo `localizacion`, mientras que si el valor es *consumible* busca los atributos `vida`, `fuerza`, `destreza`, `inteligencia` y `codicia`. Independientemente del valor de `tipo`, siempre buscará los atributos `nombre` y `descripcion`. En caso de duda consultar el cuadro de Código 8.1

Código 8.1: Formato XML de los objetos

```
<?xml version="1.0" encoding="UTF-8"?>
<Objetos>
   <tipo tipo="clave">
      <objeto nombre="[NombreIdentificativo]"
      descripcion="[Descripción]"
      localizacion="[NombreLocalización]"/>
   </tipo>
   <tipo tipo="consumible">
      <objeto ombre="[NombreIdentificativo]"
      descripcion="[Descripción]"
      vida="[ValorEntero]"
      fuerza="[ValorEntero]"
      destreza="[ValorEntero]"
      inteligencia="[ValorEntero]"
      codicia="[ValorEntero]"/>
   </tipo>
</Objetos>
```

Capítulo 9

Simulaciones de historias

En este capítulo se van a exponer algunos de los resultados que se han obtenido durante la elaboración del proyecto. Con estos resultados expuestos lo que se quiere mostrar es el alcance real del sistema y los cambios que se pueden producir cambiando los archivos del SCD.

9.1. Flujo de una historia

Como se ha mencionado anteriormente, el hilo principal de la historia es el secuestro de la *Víctima* por parte del *Secuestrador*. Pero lo que se ha introducido son una gran cantidad de factores que hacen que cada simulación el nudo y el desenlace de la historia sea diferente. Todo esto se consigue gracias al haber introducido elementos en la historia que permiten generar aleatoriedad dentro de las mismas acciones. Finalmente si se cree que el número de historias diferentes generadas no es suficiente para eso esta todo el SCD que se ha creado.

9.2. Historias en función de la parametrización

Con el SCD se ha conseguido que si el número de historias que se genera con la parametrización principal no parece suficiente o mismamente uno se ha cansado del ámbito en el cual se desarrollan, se pueden generar nuevos tipos de historias con unos pequeños ajustes. Para llevar a cabo el cambio del entorno de la simulación simplemente hay que cambiar los siguientes archivos XML(cuadro de Código correspondiente): "Antagonistas.XML"(7.2), "Frases.XML"(7.5),"Mapa.XML"(6.1),"Objetivos.XML"(7.6),"Objetos.XML"(8.1), "PNJs.XML"(7.3), "Protagonistas.XML"(7.1) y "Razas.XML"(7.4).

También se recomienda cambiar las imágenes que se muestran a través de la interfaz para conseguir un efecto más apropiado para la ambientación de estas nuevas historias. Las imágenes que se recomienda cambiar son tres: "MapaBeta.PNG", "miniMapa.PNG", "OnePage.PNG". La imagen "MapaBeta.PNG" es la que se muestra cuando a través del menú se accede a Mapa>Mostrar Mapa y es donde se muestra todas las regiones por la cuales se desarrolla la historia generada, mientras que "miniMapa.PNG" es la imagen que se carga al ejecutar el sistema y en todo momento esta visible, para la configuración principal del sistema la imagen que se muestra es un mapa más global que hace referencia a los reinos que existen. Finalmente la imagen "OnePage.PNG" es la que sirve de fondo sobre el cual se escriben las historias generadas.

9.2.1. Historias con parámetros mínimos

Para las historias más reducidas que se pueden generar se necesita que en el archivo "Protagonistas.XML" haya como mínimo un personaje *Víctima*, un personaje *Allegado* a esta víctima y un *Aspirante*, mientras que en el archivo "Antagonistas.XML" solo tiene que haber un personaje *Secuestrador*. Creando un mapa con dos localizaciones es más que suficiente para que se genere la historia. Y por último se necesita el archivo "Frases.XML" que este será el más extenso de todos. Para crear esta historia mínima no son necesarios los archivos "Razas.XML", "PNJ.XML"y "Objetos.XML".

9.2.2. Ampliando parámetros

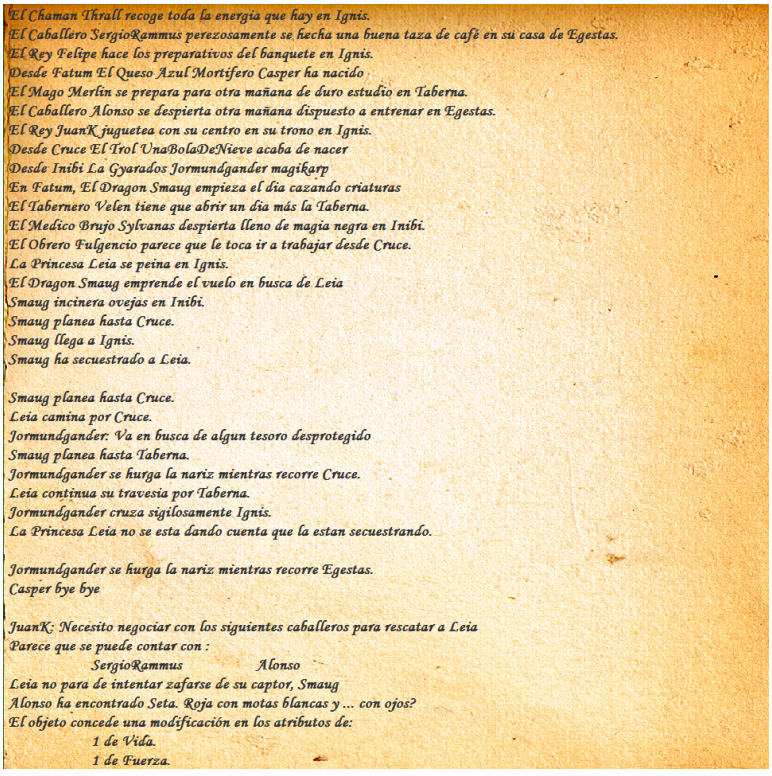
Partiendo desde la historia reducida, simplemente incluyendo los archivos que no eran necesarios ya se consiguen nuevas historias o ampliaciones de las historias reducidas. Por tanto a cuanto mayor cantidad de datos se introduzcan en los archivos XML que afectan a la generación de la historia, la cantidad de historias diferentes que pueden conseguir crece de forma exponencial. La siguiente historia se ha generado con una configuración mínima de al menos un tipo de cada personaje descrito en el Capítulo 7, además de introducir un mapa extenso para evitar que al generar una historia se produzcan los mismos encuentros. También en esta historia se puede ver la interactuación de los personajes con los objetos y como les afectan. Por último es importante saber que toda la narrativa de esta historia es la que se encuentra almacenada y simplemente modificando este almacenaje cambiaría.

9.3. Ejemplos de historias

A continuación se va a mostrar ejemplos de historias que ha generado el sistema. Todas las historias funcionan gracias al intercambio de mensajes entre los Agentes. Para ver ejemplos de estos intercambios hay que mirar el Apéndice A.

9.3.1. Una Historia

En este primer ejemplo se va a mostrar una de las ejecución del sistema, la cual se encuentra enmarcada en un ambiente medieval fantástico. En esta historia se observa en primer lugar donde se encuentran todos los personajes que van a participar en la historia. Acto seguido empieza a desarrollarse la trama en el momento que el dragón emprende la marcha para ir a buscar a la princesa. Una vez esta secuestrada la princesa, el captor vuelve a la guarida junto a su presa. En paralelo el padre de la princesa empieza una búsqueda por el reino para ver que caballeros están disponibles para rescatarla. Este fragmento de historia se puede ver en la Imagen 9.1.


El Chaman Thrall recoge toda la energia que hay en Ignis.
El Caballero SergioRammus perezosamente se hecha una buena taza de café en su casa de Egestas.
El Rey Felipe hace los preparativos del banquete en Ignis.
Desde Fatum El Queso Azul Mortifero Casper ha nacido
El Mago Merlin se prepara para otra mañana de duro estudio en Taberna.
El Caballero Alonso se despierta otra mañana dispuesto a entrenar en Egestas.
El Rey JuanK juguetea con su centro en su trono en Ignis.
Desde Cruce El Trol UnaBolaDeNieve acaba de nacer
Desde Inibi La Gyarados Jormundgander magikarp
En Fatum, El Dragon Smaug empieza el dia cazando criaturas
El Tabernero Velen tiene que abrir un dia más la Taberna.
El Medico Brujo Sylvanas despierta lleno de magia negra en Inibi.
El Obrero Fulgencio parece que le toca ir a trabajar desde Cruce.
La Princesa Leia se peina en Ignis.
El Dragon Smaug emprende el vuelo en busca de Leia
Smaug incinera ovejas en Inibi.
Smaug planea hasta Cruce.
Smaug llega a Ignis.
Smaug ha secuestrado a Leia.

Smaug planea hasta Cruce.
Leia camina por Cruce.
Jormundgander: Va en busca de algun tesoro desprotegido
Smaug planea hasta Taberna.
Jormundgander se hurga la nariz mientras recorre Cruce.
Leia continua su travesia por Taberna.
Jormundgander cruza sigilosamente Ignis.
La Princesa Leia no se esta dando cuenta que la estan secuestrando.

Jormundgander se hurga la nariz mientras recorre Egestas.
Casper bye bye

JuanK: Necesito negociar con los siguientes caballeros para rescatar a Leia
Parece que se puede contar con :
SergioRammus Alonso
Leia no para de intentar zafarse de su captor, Smaug
Alonso ha encontrado Seta. Roja con motas blancas y ... con ojos?
El objeto concede una modificación en los atributos de:
1 de Vida.
1 de Fuerza.

Figura 9.1: Imagen del ejemplo de una simulación(1)

En la siguiente Imagen, 9.2, lo que se está narrando es como Alonso a sido el caballero escogido y emprende su marcha para cumplir el objetivo que le ha pedido el rey. Durante su travesía se observa como se encuentra con un PNJ, hace una paradita para comer una manzana y se encuentra con simpático emboscador que se encarga de sacarle el dinero. En último lugar se ve el enfrentamiento entre Alonso y el dragón siendo este derrotado.

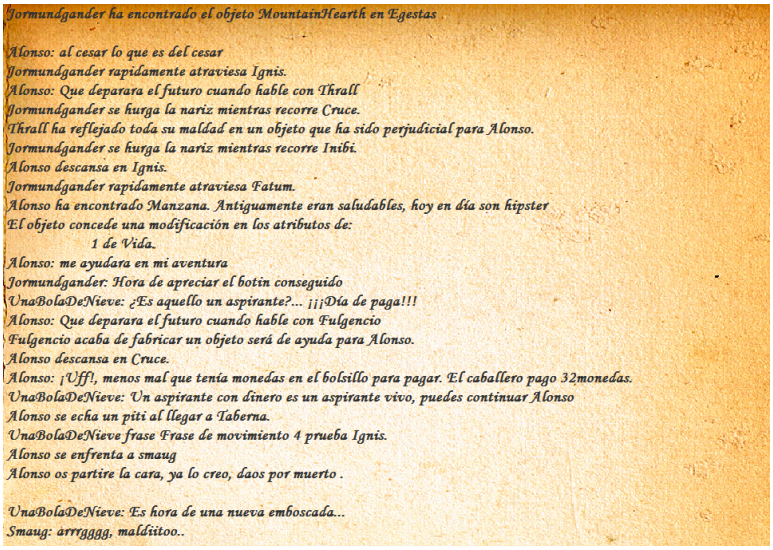


Figura 9.2: Imagen del ejemplo de una simulación(2)

Una vez el caballero se encuentra con la princesa, este en un acto de absoluta maldad decide atacar a la princesa, lo que le convierte en un villano. Este acto ocasiona una lucha entre ambos donde hay un final trágico, ya que mueren los dos personajes. Al morir la princesa se produce un gran pesar en todo el reino y se puede ver como el otro caballero estaba enamorado de la princesa porque esta ahogando sus penas. La Imagen 9.3 hace referencia a lo explicado.

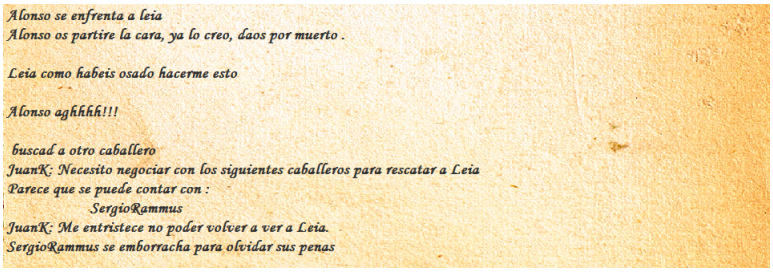


Figura 9.3: Imagen del ejemplo de una simulación(3)

9.3.2. Mismo entorno, distinto final

Dentro de este mismo encuadre se va a mostrar un ejemplo (9.4) en el que no tiene porque siempre la princesa esperar a un caballero que la rescate de las garras del dragón, sino que ella es una mujer fuerte e independiente capaz de huir sigilosamente de la guarida de su captor y llegar a su querido hogar para finalizar así la historia.

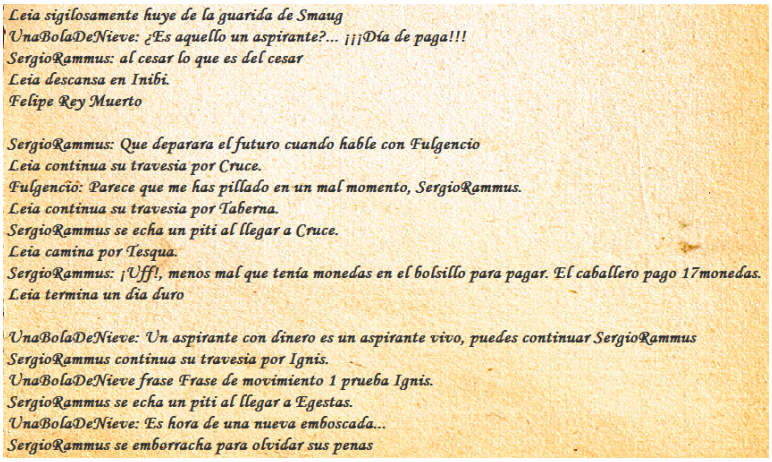


Figura 9.4: Imagen del ejemplo de una simulación(4)

Estos ejemplos representan posibles transcursos de una misma historia, donde los personajes que intervienen en ella, los caminos e incluso los finales divergen. De este modo, se demuestra que este sistema es capaz de generar historias diversas dentro de un mismo entorno.

9.3.3. Historias famosas

Además de conseguir multitud de historias también el sistema permite recrear algunas historias conocidas, como la historia de *Super Mario*, *Zelda* o *Shrek*, simplemente introduciendo los datos en los archivos del SCD. En la siguiente imagen se puede observar una pequeña historia sobre basada en algunos de los personajes de la franquicia del videojuego Super Mario. El resultado obtenido ha sido posible realizando las pautas que se han explicado en la Sección 9.2 dentro de este mismo Capítulo. Con esto se quiere demostrar que el sistema es capaz de realizar integraciones de ámbitos distintos, de esta manera sería sencillo imitar ambientaciones ya existentes como crear una ambientación nueva y después trabajar sobre ella.

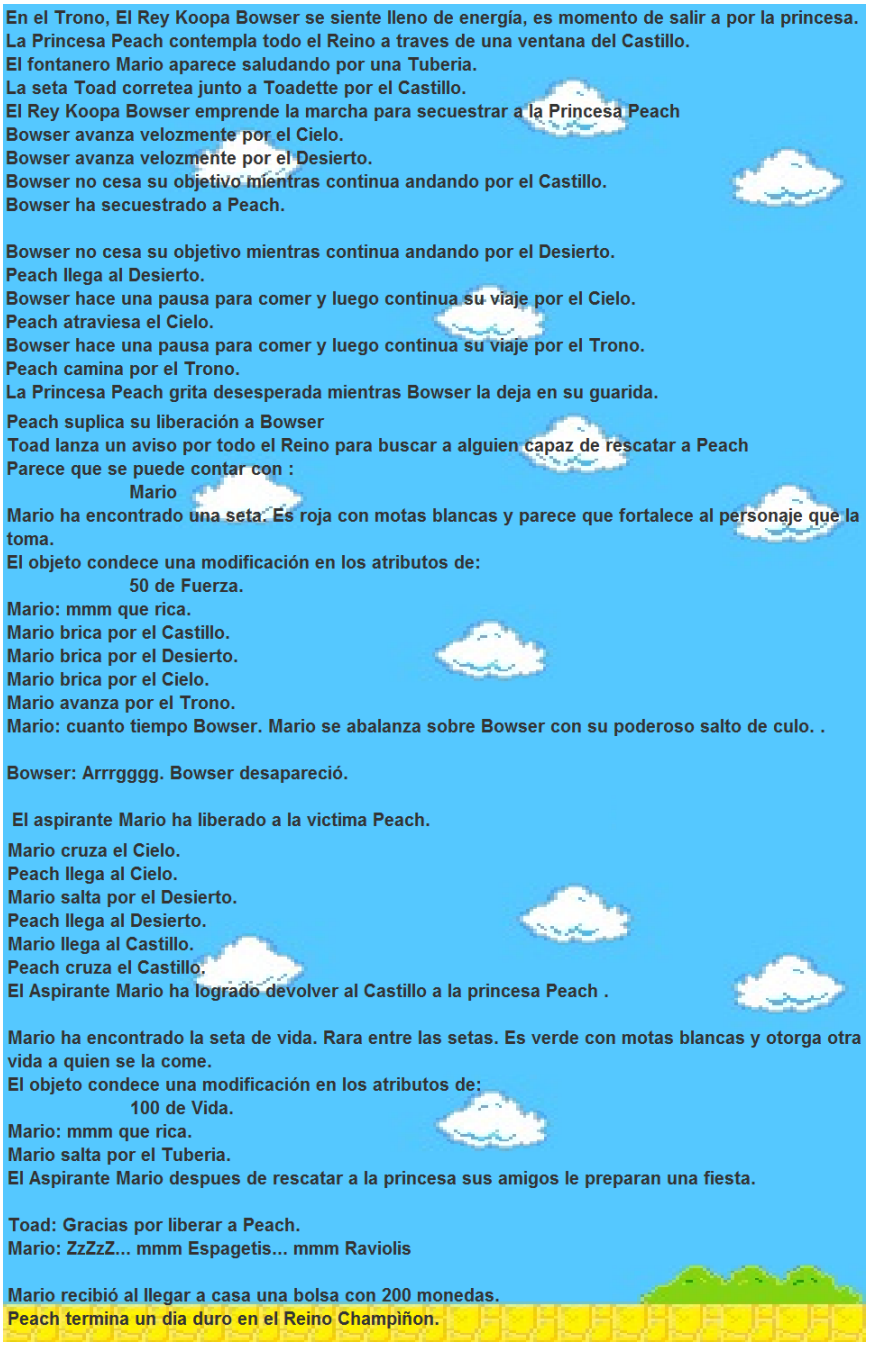


Figura 9.5: Imagen de una simulación con parámetros de Super Mario

Capítulo 10

Trabajo Individual

Antes de empezar es importante señalar que el trabajo hecho para el desarrollo de esta investigación ha sido realmente extenso. Si bien hay ciertas partes que se pueden dividir en tareas que ha realizado de manera individual cada uno de los miembros, es importante señalar que sin la ayuda de todos no se habrían conseguido cotas tan altas. Las reuniones de contenidos, opiniones, revisiones y modificaciones son difíciles de detallar para explicar de manera precisa donde acaba la aportación de un integrante y empieza la de otro. A continuación se procederá a intentar explicar de la mejor manera posible las aportaciones individuales de cada uno, teniendo siempre en cuenta esta premisa.

10.1. Trabajo realizado por Juan Orellana Quemada

Para empezar mi aportación a esta investigación me documente bien sobre la misma, pues nunca había trabajado ni con generadores de historias ni con agentes inteligentes ni con un planificador. Para ello, y como se partía de una investigación anterior, tuve que leer profundamente la memoria de la misma así como los artículos referenciados en ella, haciendo especial hincapié en aquellos que nos proporcionaba nuestro director del proyecto.

A partir de aquí, me volqué en el diseño de nuestro generador de historias. Como no teníamos muy claro que queríamos hacer en un principio, participé activamente en el *brainstorming* inicial, y poco a poco conseguimos buenas ideas junto a la ayuda de nuestro tutor, detalladas mas extensamente en el Capitulo de 3.

Como tampoco podíamos paralizar la implementación, en cuanto tuvimos un objetivo claro, en este caso el de los personajes con múltiples objetivos, realice las primeras pruebas de modificaciones en la aplicación, hasta que al no conseguir avances hubo que rehacerla, tal y como se detalla en el Capítulo 4. En concreto todas las pruebas del villano fueron realizadas por mi.

Mientras intentaba esto, también desarrollé el primer prototipo de memoria de la aplicación, de forma que aquellas partes en las que el diseño era claro iban tomando forma poco a poco dando lugar a unos objetivos claros hacia los que encaminar nuestro trabajo. Sin embargo hablaré del resto de mi aportación a la memoria más tarde.

Cuando empezó la implementación de nuestra aplicación definitiva, aprendí de mi compañero Oscar los fundamentos de la misma, pues él llevaba la voz cantante en este aspecto. Cuando adquirí el suficiente conocimiento, implementé la primera beta de la acción `Batalla`, y construí el SCD de los PNJ's.

Es a partir de este momento cuando decidimos que queremos una aplicación altamente configurable, y es en este apartado donde he realizado mi principal aportación al proyecto.

Lo primero que intente fue tratar de hacer a los personajes lo mas configurables posibles. Para ello, debí crear una nueva estructura para los propios personajes, a través de una superclase y el uso de la herencia (el resultado final de esta arquitectura se puede ver en la Figura 7.1) . Para ello, fue necesario crear las clases `Protagonista` y `Antagonista`, de la misma manera que tuve que rehacer `PNJ`, de manera que todas las clases pudiesen interpretar correctamente los datos de la superclase. Estos cambios en la estructura me permitieron configurar los personajes de una manera más eficaz, de manera que les añadí atributos para favorecer la narración, como por ejemplo en un antagonista su `sexo` o su `especie`. Esto, junto al atributo `clase`, me permitía configurar los detalles de un antagonista, de manera que con un mismo arquetipo de configuración conseguía resultados muy distintos, como por ejemplo los que muestro en el Cuadro 10.1.

Código 10.1: Ejemplo de código de creacion de antagonistas

```
<Actores>
   <monstruo nombre="Smaug" clase="Secuestrador"
      especie="Dragon" sexo="M"/>
   <monstruo nombre="Florecilla" clase="Secuestrador"
       especie="Dragon" sexo="F"/>
   <monstruo nombre="Scar" clase="Secuestrador"
       especie="Tigre" sexo="M"/>
</Actores>
```

De esta manera, puedo conseguir al dragón Smaug, la dragona Florecilla o el tigre Scar, y todos actuaran como un antagonista secuestrador. La gracia de esta implementación reside en que la historia cambia totalmente cuando

se narra, gracias a los atributos nuevos y la abstracción con la que se han construido las clases de personajes.

Cuando implementé estos cambios, se hizo patente la necesidad de cambiar los archivos XML y los SCD que los cargaban. De esta manera, configuré correctamente el SCD, lo cual provocó que se debía cambiar la manera en la que se iniciaba cada personaje de manera individual y tuvé que realizar los cambios oportunos.

Mi siguiente trabajo fue cambiar el mapa, tratando de permitir al usuario modificarlo a su gusto. Al rehacer el mapa de la manera mas configurable posible decidí darle tipos a las localizaciones, consultable en el Capítulo 6.2.2, de manera que los personajes se situaban al crearse según su clase en estos distintos tipos, tal y como se muestra en la Figura 6.1.2. Todas estas características están implementadas en la interfaz `Vocabulario`, donde se definen los diferentes tipos de localizaciones que se pueden tener, así como en la clase `Mitologia` donde se especifica la relación entre cada clase y su localización. Para conseguir satisfactoriamente este cambio, tuve que crear una arquitectura funcional para el mismo, creando la clase localización para poder interpretar correctamente el mapa y revisando la conexión entre la clase `Mapa` y el `Agente Mundo`, de manera que este agente pudiese interpretar correctamente el mapa (el diagrama de esta arquitectura es consultable en la figura 6.5).

Lo único que quedaba sin ser configurable en la aplicación eran las razas, así que me puse con ello. Originalmente, las razas se cargaban directamente desde un enumerado implementado dentro de la aplicación, de manera que para hacerlas configurables tuve que crear su propia clase, asi como modificar su XML y crear un SCD acorde a él. De igual manera, tuve que modificar la clase `Protagonista` para que las razas fuesen cargadas correctamente y hacer que esta clase pudiese interactuar correctamente con la clase raza, conseguido a través de su vínculo común con el `Agente Mundo`.

Por último realice la implementación del `Ayudante del Héroe`. Para crear este agente, debí crear su clase, así como los métodos necesarios para su ejecución y crear los comportamientos que necesitaba, en este caso `ConvertirseEnSabio`. Sin embargo, la parte mas difícil de esta implementación fue conseguir que el mundo avisase correctamente al `ayudante` cuando se producía un encuentro entre un `aspirante a heroe` y un `ladron`. Para ello, debí modificar el `Agente Mundo`, de manera que tomando como ejemplo el encuentro entre el `aspirante` y el `emboscador`, realicé uno parecido entre el `aspirante a heroe` y el `ladron` llamado `EncuentroLadron`, controlando como interactuaban estos agentes entre ellos, buscando en las páginas amarillas de JADE y mediante el intercambio de mensajes. Una vez se producía el encuentro entre estos dos agentes, el `Agente Mundo` se encargaba de avisar a los ayudantes a través del método `AcudeAyudante`, tal y como se especifica en el capitulo7.2.2.5, donde nuevamente controlé el intercambio de mensajes

entre estos agentes hasta que finalmente el ayudante llegaba. Por último, solo me quedaba realizar correctamente la planificación del ayudante, y junto con la ayuda de Oscar, conseguí acabarla satisfactoriamente.

Para finalizar, me comprometí en gran medida con el desarrollo y corrección de la memoria. En concreto, los capítulos que he desarrollado han sido: Resumen y su traducción, Introducción y su traducción, Primeros Pasos, Mapa, Personajes (en concreto, la clase Personaje, personajes, antagonistas, PNJ's, atributos y razas) y Conclusiones y su traducción.

10.2. Trabajo realizado por Samuel Rodríguez Oliva

En primer lugar para poder empezar a hablar de mi aportación al proyecto que se ha estado describiendo a lo largo de esta memoria hay que empezar por el trabajo de documentación que he ido realizando. Al no haber trabajado previamente con Agentes ni con planificadores, le pedí al tutor toda la información posible sobre el proyecto del cual íbamos a tener partir. Aparte de la información del proyecto también nos pasó cinco documentos que trataban sobre Agentes, Sistemas Multi-Agentes y generadores de historias con Agentes, referencias donde ver como instalar y trabajar con JADE y una web (http://icaps-conference.org/index.php/Main/Competitions), en la cual se podían ver proyectos que han participado en competiciones, para buscar información sobre planificación en IA en general y PDDL en particular. Toda esta información sirvió para focalizar la investigación en ciertos puntos que establecerían algunas base del proyecto. A partir de aquí comencé a leer artículos relacionados con la generación de historias y la web http://nil.fdi.ucm.es/ fue de gran ayuda al ser donde encontré la mayoría de ellos. Esto me permitió conocer sistemas previos de generación de historias lo que me fue de gran ayuda para conocer aspectos en los cuales ya se habían trabajado previamente además de observar y adaptar nuevas ideas que integrar en el proyecto. Para conocer los sistemas con mayor relevancia dentro del campo de la generación de historias automáticas es mejor ver el Capítulo 2.3.

Durante las primeras semanas fue crucial realizar reuniones cada dos semanas ya que en ellas Gonzalo, y durante todo el tiempo que le fue posible Raquel, nos solventaban las dudas que habían ido surgiendo de una reunión a otra al ir avanzando y trabajando sobre el proyecto. Después de la resolución de dudas, se realizaba un *brainstorming* sobre las funciones que queríamos que realizase nuestro sistema. Aunque no todas las ideas lanzadas en estas reuniones nos servían o podías ser implementas si recogimos algunas que han servido como pilares para el sistema. Estos pilares están recogidos como los objetivos que hemos querido conseguir y de los cuales se habla detalladamente en el Capitulo de 3.

Como se ve en el Capítulo 4 estábamos bloqueados con la implementación ya que nos atamos, al principio, con el sistema de nuestros compañeros. En estos primeros pasos trabajé en como hacer que la princesa no fuese un Agente objeto y ayudé a mis compañeros para intentar lograr que el caballero pudiese tener múltiples objetivos. Al fracasar en estos objetivos decidimos partir de un sistema realizado por nosotros desde el principio y así no tener limitaciones que no fuesen impuestas por nosotros mismos. En este punto repartimos el trabajo para avanzar de forma más eficiente y es aquí donde yo empecé a trabajar con el sistema finalmente desarrollado.

En primer lugar mi trabajo se centro en la inclusión de los objetos en el sistema para que los personajes pudiesen interactuar con ellos afectando el transcurso de la historia y dando lugar a nuevas variantes. Para ver más detalladamente este apartado consultar el Capítulo 8. Ya que para algunos objetos, los clave(Capítulo 8.1.2), necesitaba trabajar con la parte de planificación, hubo momentos en los que trabajamos conjuntamente Óscar y yo, él ayudándome con la integración de los objetos en el planificador y yo a él con el archivo de dominio del planificador en la creación de algunas acciones atómicas de los personajes. A la vez que trabajaba con los objetos también estaba participando en la realización de conceptos, tal cual se encuentran implementados actualmente, como los atributos y razas, dando lógica a todo este nuevo apartado. Además también participe activamente en la implementación del árbol de herencia de clases de los personajes.

Una vez implementado la primera versión estable de los objetos, mi trabajo se centró enteramente al apartado narrativo del sistema. Dentro de este apartado es importante explicar que el porqué no tiene un Capitulo propio dentro de la memoria y es que he creído conveniente que la mejor opción para explicar el tema sería en un apartado dentro del Capítulo de Personajes(7.6), debido a que las historias generadas con nuestro sistema están centradas en los personajes y son estos los encargados de saber para cada acción que realizan la frase o frases correspondientes que pueden decir durante la consecución de sus respectivos objetivos.

En los trabajos que acabo de describir también me he encargado de la realización de su correspondiente SCD. Es por ello que esto me ha servido para poder estar encargado de la realización de las nuevas configuraciones para los archivos XML y de imágenes con los que trabaja el sistema, y con ello poder probar el potencial de configuración y cambio de entornos de nuestro sistema. Para ver una explicación más detallada y algún ejemplo de historias generadas es conveniente mirar en el Capítulo 9. El ir probando estas diferentes configuraciones también ha servido para haberme dedicado al control de errores y haber ido solventando los problemas y errores que se encontraban en las últimas versiones del sistema.

De forma paralela a la implementación del sistema, en primer lugar me encargué de transferir al documento que empezamos a generar en *LaTeX* toda la información que hasta el momento habíamos ido recogiendo tanto yo como mis compañeros. Aquellos aspectos sobre los cuales he escrito han sido referentes principalmente a la parte de Narrativa Computacional dentro del Estado de la Cuestión, al Capítulo sobre los objetos así como al Capítulo de Simulaciones realizadas y al aparatado del Sistema Narrativo de los Personajes, que son en los que se ha centrado mi trabajo de implementación. También he participado de forma importante en el desarrollo del Resumen, Introducción y Trabajo Futuro, además de en el replanteamiento y la reestructuración de los Capítulos Primeros Pasos y Personajes. Finamente mis últimos pasos en la realización de esta memoria se han enfocado en la corrección, de la forma más eficiente posible, de todos los Capítulos, en especial sobre los que no había trabajado, además de una correcta colocación de imágenes, textos y códigos que se han servido tanto de ejemplo como de apoyo durante la descripción del proyecto en la redacción de este documento.

10.3. Trabajo realizado por Oscar David Vásquez Peraza

Una vez que se realizaron los primeros procesos de *brainstorming*, quedó bastante claro que las primeras aproximaciones que se debían hacer era familiarizarse con JADE, el lenguaje PDDL, y estudiar al fondo el sistema generador anterior.

En estas primeras fases, se trató de incorporar una serie de variaciones y nuevas funciones en el sistema base, sin mucho éxito. Entonces empecé a crear sistemas simples usando JADE que ayudarían al resto de compañeros también a entender el funcionamiento de los agentes con mayor facilidad. Una vez hecho todo esto procedí con la reimplementación del sistema anterior parte por parte para poder tener una estructura limpia que nos permitiese realizar la implementación de las nuevas funciones. De igual manera, he sido responsable de crear y mantener la consistencia de los repositorios.

Tras varias discusiones sobre las diferentes capas que iban a componer el sistema, se empieza a hacer el montaje del nuevo sistema. Una de las primeras funciones que se incorpora es la interfaz gráfica, puesto que nos permitiría interactuar con el sistema de una forma más cómoda. De primeras, la función principal de la interfaz gráfica fue disponer de una única ventana que tuviese un área de texto que sustituiría a la salida estándar de consola que estaba usando la aplicación por defecto. A medida que se fue avanzando, se estudiaron diferentes diseños para la interfaz, de los cuales el que se usa trata de ser un diseño limpio y sencillo que permita centrar la atención en el cuadro central donde se muestra la historia y minimizar otros componentes extras que puedan distraer al usuario de la historia que se está generando.

El diseño e implementación de nuevos tipos de personajes requería no solo un buen manejo de JADE, sino que tambíen del lenguaje PDDL puesto éste es quién declara y especifica las acciones que usan los personajes y que en función de las acciones que realicen serán un tipo de personaje en la historia. Todo esto lleva a empezar a diseñar el conjunto de instrucciones necesarias para disponer de un sistema multiobjetivos.

La solución más sencilla con la que dimos fué la de permitir a los archivos de configuración disponer de varias entradas para las etiquetas XML e implementar del lado de los SCD una carga aleatoria de las mismas. Esto obliga a que la implementación de cada tipo de personaje sea lo suficientemente robusta como para aceptar distintos comportamientos cuya ejecución vendría determinada por la secuencia de acciones que sería devuelta por el planificador de la aplicación. En este momento nos damos cuenta de que en lugar de crear un agente en concreto que solo realizaba cierto tipos de acciones, se puede hacer que el agente sea capaz de ejercer ciertos roles en función del tipo de objetivo que haya elegido realizar.

Así es como se empieza a hacer la ampliación del dominio pddl y crear nuevas acciones, con nuevos efectos sobre los personajes y que algunos servirían para personajes específicos y otras acciones más generales. Esto obliga ampliar el código para que cada personaje sea capaz de procesar las nuevas instrucciones que va recibiendo del planificador así como también aumentó drásticamente el número de variables de control en el sistema.

A la vista de la nueva magnitud que el sistema estaba tomando, fue necesario modificar el agente Mundo y la forma en la que manejaba cierta información. Debido a ciertas características del lenguaje pddl, la información que recibe cada personaje para generar su fichero de problema empezó a verse en la necesidad de ser filtrada, puesto que el tamaño de los archivos ha aumentado porque recogen toda la nueva información. Gran parte de esta información era irrelevante para cada personaje, o mejor dicho, a cada personaje sólo le interesa conocer cierta información, así que generar un archivo de problema pddl específico para cada personaje se convirtió en algo a implementar con alto nivel de urgencia.

El montaje del nuevo sistema permitía que implementar una serie nuevos personajes fuese más sencillo, empezando a implementar personajes como el emboscador, el asesino, el ladrón, la planificación de la víctima , así como darle la posibilidad de escapar por cuenta propia de su secuestrador.

Al implementar a estos personajes, crear y diseñar la forma en la que se dispararían eventos entre personajes que estuviesen funcionando de forma paralela se hace inmediatamente necesario para que estos personajes interactúen correctamente, con lo cuál me hago cargo de asegurarme que el agente Mundo sea capaz de manejar y controlar los eventos. A modo general, la mayor parte de mi trabajo está centrado en el estudio y diseño del funcionamiento y la lógica del sistema.

Respecto a la documentación, al ser el único integrante del grupo que tenía experiencia previa en LaTeX, me encargué al principio de hacer el montaje y pruebas de la documentación sobre la plantilla TeXiS de igual manera enseñar a mis compañeros la sintaxis e instrucciones básicas que usaríamos con mayor frecuencia en la redacción de éste documento.

En la redacción, los capítulos de los cuales he tenido mayor responsabilidad han sido: Estado de la Cuestión (a excepción de la narrativa), Objetivos, Arquitectura y Trabajo Futuro. De igual manera al resto de compañeros, responsable de la revisión y corrección sistemática de cada uno de los diferentes capítulos de este documento.

Capítulo 11

Conclusiones

Como bien se ha comprobado durante el desarrollo de esta investigación, la generación automática de historias es un campo muy complejo y extenso, en el que cualquier nueva vertiente añade un abanico casi infinito de posibilidades nuevas para agregar a la investigación. Al querer cumplir los objetivos propuestos para el sistema, la investigación se introduce en numerosas ramas, provocando que se corra el riesgo de abarcar demasiado y no llegar a los objetivos propuestos.

Con el suficiente cuidado para evitar este problema, esta investigación ha conseguido añadir numerosos personajes complejos a la narración, donde todas sus acciones tienen repercusión en la misma. Estos personajes interactúan entre ellos, provocando que las acciones de un personaje desencadenen la aparición de otro o provoquen un cambio en su manera de actuar. Así mismo, sus diferentes rasgos y características les hacen únicos, haciendo que la única manera posible de hacer dos personajes iguales es crearlos con ese propósito.

De la misma manera, el entorno ha conseguido un papel más determinante en la historia, provocando que los lugares que finalmente atraviesa un personaje durante su ejecución modifiquen el resultado de la historia, al consumir el personaje los objetos escondidos en dichos lugares o provocando que se interactúe con los personajes ligados a éstos.

Gracias a estos cambios se consigue que los personajes sean poco predecibles y que muy raras veces actúen de la misma manera en dos simulaciones consecutivas o consigan los mismos resultados.

También se ha conseguido una aplicación altamente configurable donde el usuario es capaz de narrar de una manera más o menos eficiente la historia que él a propuesto. De esta manera, se ha conseguido personalizar la genera-

ción automática de historias, de manera que ya no tienen por que generarse diferentes historias sobre el escenario propuesto, si no que el usuario es capaz de definir su propio entorno y centrar la generación de historias automática en torno a *su* historia.

¿Se puede decir por tanto que se han cumplido los objetivos?
Si, se ha conseguido crear un generador de historias automático capaz de generar una gran cantidad de historias variadas, con infinitas posibilidades de personalización que a su vez generan mayor diversidad a la hora de crear historias. Pero ni mucho menos la investigación sobre la generación automática de historias ha acabado. Esta misma investigación a abierto nuevas variantes para tratar la diversidad en la generación o intentar emular la creatividad en la inteligencia artificial. Algunas de estas variantes son expuestas en el siguiente capítulo.

Conclusions

As has been proved throughout the development of this research, automated storytelling is a very complex and vast field. Every new aspect on the research adds an almost infinite new number of possibilities. In order to fulfil the objectives of the research, numerous new branches are introduced, producing the risk of not reaching the proposed objectives.

Trying to avoid this problem, this research adds many complex characters to the story, and all their actions have an impact on the story. These characters interact between them, causing with their actions the appearance of another character or a change in the way they act. Any of these characters has their unique characteristics, which means that the only possible way of making two identical characters is to create them with that purpose.

Similarly, the environment has got a more decisive role in the story, causing that the places a character finally crosses across its execution modify the story because this character will consume the hidden objects in such places or will interact with characters on them .

Thanks to these changes the characters are unpredictable and rarely act in the same way in two consecutive simulations or get the same results.

This way, a highly configurable application where the user is able to tell the story he proposed has been implemented. On the other hand, we managed to customize the automated storytelling, so the user has no longer to generate different stories on the proposed environment but he is able to define his own environment and tell *his* stories.

So, are we able to confirm that the targets have been reached??
Yes, an automated storytteller capable of generating a lot of varied stories has been developed, with infinite possibilities of personalization which helps a lot generating diversity in the stroytelling. But the research on this field is far to be over. This same project has open a lot of new variants trying to emulate creativity on an artificial intelligence. Therefore, some of these variants are exposed on the following chapter.

Capítulo 12

Trabajo Futuro

- Pero Doc, ¿has construido una máquina del tiempo con un DeLorean?
-En mi opinión, si vas a hacer algo como esto, hazlo con estilo.

Conversación entre Doc y Marty,
Regreso al futuro

Con el fin de progresar y ampliar las funcionalidades actuales del sistema se proponen varios puntos de investigación para mejorar el mismo.

12.1. Personalidad y complejidad de los personajes

Uno de los aspectos que más ha influido en aumentar la variedad de las historias generadas ha sido la inclusión de nuevas acciones disponibles. Cuanto mayor sea la cantidad de acciones que pueden realizar los personajes durante su ejecución mayor cantidad de historias habrá. Es por ello fundamental el que existan más acciones atómicas, con las que se podrán realizar acciones aún más complejas.

Dentro del mismo apartado de personajes, agregar aspectos como la personalidad o estados de animo (miedosos, valientes, introvertidos...) permitirá que este aspecto influya tanto en las frases que dicen los personajes como en la probabilidad de realizar con éxito ciertas acciones.

Por último, los atributos y razas tomados como un factor aleatorizante en el transcurso de las historias abre un amplio abanico de posibilidades. Incorporar al sistema eventos o limitaciones ligados a estas características enriquecería enormemente al sistema.

12.2. Estabilidad

El aumento en la capacidad de configuración del sistema ha obligado a perder control automático en la cohesión de las diferentes fuentes que proporcionan datos a la aplicación. Para hacer que el sistema sea más robusto surgirá la necesidad de tener herramienta encargada de la correcta cohesión y consistencia entre las diferentes fuentes de todo el SCD. De esta forma se conseguirá aumentar la sencillez y comodidad a la hora de configurar y usar el sistema por parte de cualquier usuario.

12.3. Interfaz Gráfica de Usuario

La interfaz gráfica actual que usa el sistema es lo suficientemente sencilla para dejar incorporar nuevos personajes a la historia en ejecución pero se ve muy limitada para otras funciones. Extender la implementación de la interfaz para que funcione como cuadro de mandos y permita al usuario introducir los datos iniciales para el sistema en lugar de tener que preparar los archivos XML con antelación facilitaría el uso de la aplicación.

Así como disponer de acciones que permitan pausar o poder modificar el estado actual de un personaje mientras la historia se está generando permitiría aumentar la interactividad general del usuario contra el sistema.

12.4. Agentes y Sincronización

Actualmente, la mayoría de los agentes pasan buena parte del tiempo esperando a que ocurra algún evento, mientras esto pasa, los agentes no realizan ninguna acción. Incorporar comportamientos a los agentes para minimizar el tiempo que pasan ociosos y que aporten nueva información a las historias.

En general, aunque existen eventos que se disparan cuando dos personajes se llegan a encontrar, es necesario que para que un primer agente cumpla su planificación, el resto de agente con los que interactúa estén en un estado de *standby*. Implementar una interacción por parte del usuario en tiempo de ejecución requiere que los agentes sean capaces de actualizar la información que conocen del entorno en cada momento, cosa que actualmente no realizan en totalidad.

12.5. Hilo desencadenante de la historia

En el momento actual el desencadenante de la historia siempre es que el Secuestrador rapta a la Princesa y su Allegado pide ayuda. Este proyecto se ha enfocado ha generar multitud de historias a partir de ese momento. Por esto queda pendiente cambiar el desencadenante de la historia para conseguir que además de generar historias diversas las haría impredecibles desde el principio.

Apéndice A

Secuencia de intercambio de mensajes

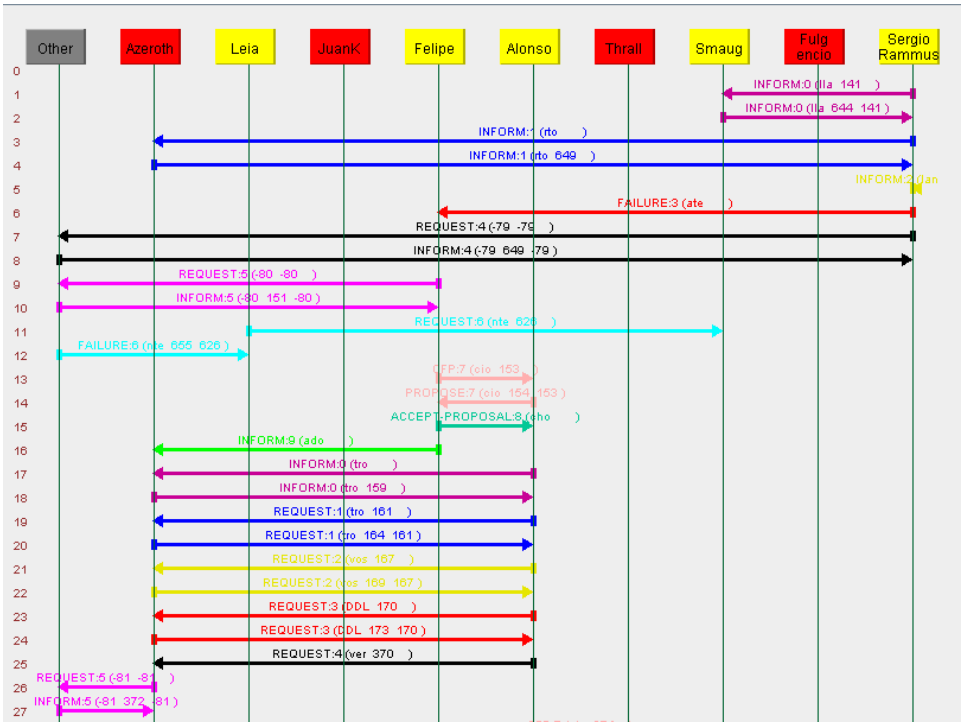


Figura A.1: Diagrama de ejemplo de intercambio de mensajes 1

Código A.1: Extracto secuencia mensajes FIPA

```
(INFORM
:sender ( agent-identifier :name Smaug@147.96.125.224:1099/JADE
:addresses (sequence http://TheRoad:7778/acc ))
:receiver (set ( agent-identifier
   :name SergioRammus@147.96.125.224:1099/JADE
   :addresses (sequence http://TheRoad:7778/acc )) )
:content "110"
:reply-with SergioRammus@147.96.125.224:1099/JADE1472472185644
:in-reply-to batalla1472472185141 :conversation-id Batalla )
(INFORM
:sender ( agent-identifier :name SergioRammus@147.96.125.224:1099/JADE
:addresses (sequence http://TheRoad:7778/acc ))
:receiver (set ( agent-identifier
   :name Azeroth@147.96.125.224:1099/JADE
   :addresses (sequence http://TheRoad:7778/acc )) )
:content "SergioRammus"
:conversation-id Muerto )
(INFORM
:sender ( agent-identifier :name Azeroth@147.96.125.224:1099/JADE
   :addresses (sequence http://TheRoad:7778/acc ))
:receiver (set ( agent-identifier
   :name SergioRammus@147.96.125.224:1099/JADE
   :addresses (sequence http://TheRoad:7778/acc )) )
:reply-with SergioRammus@147.96.125.224:1099/JADE1472472185649
   :conversation-id Muerto )
(INFORM
:sender ( agent-identifier :name SergioRammus@147.96.125.224:1099/JADE
:addresses (sequence http://TheRoad:7778/acc ))
:receiver (set ( agent-identifier
   :name SergioRammus@147.96.125.224:1099/JADE
   :addresses (sequence http://TheRoad:7778/acc )) )
:conversation-id Fin-Plan )
(FAILURE
:sender ( agent-identifier :name SergioRammus@147.96.125.224:1099/JADE
:addresses (sequence http://TheRoad:7778/acc ))
:receiver (set ( agent-identifier
   :name Felipe@147.96.125.224:1099/JADE
   :addresses (sequence http://TheRoad:7778/acc )) )
:conversation-id Rescate )
(REQUEST
:sender ( agent-identifier :name SergioRammus@147.96.125.224:1099/JADE
:addresses (sequence http://TheRoad:7778/acc ))
:receiver (set ( agent-identifier :name df@147.96.125.224:1099/JADE
   :addresses (sequence http://TheRoad:7778/acc )) )
:content "((action ( agent-identifier
   :name df@147.96.125.224:1099/JADE
   :addresses (sequence http://TheRoad:7778/acc ))
      (deregister (df-agent-description :name ( agent-identifier
      :name SergioRammus@147.96.125.224:1099/JADE
      :addresses (sequence http://TheRoad:7778/acc ))))))"
:reply-with rw-SergioRammus@147.96.125.224:1099/JADE1472472186648-79
:language fipa-sl0 :ontology FIPA-Agent-Management
   :protocol fipa-request
:conversation-id
   conv-SergioRammus@147.96.125.224:1099/JADE1472472186648-79 )
```

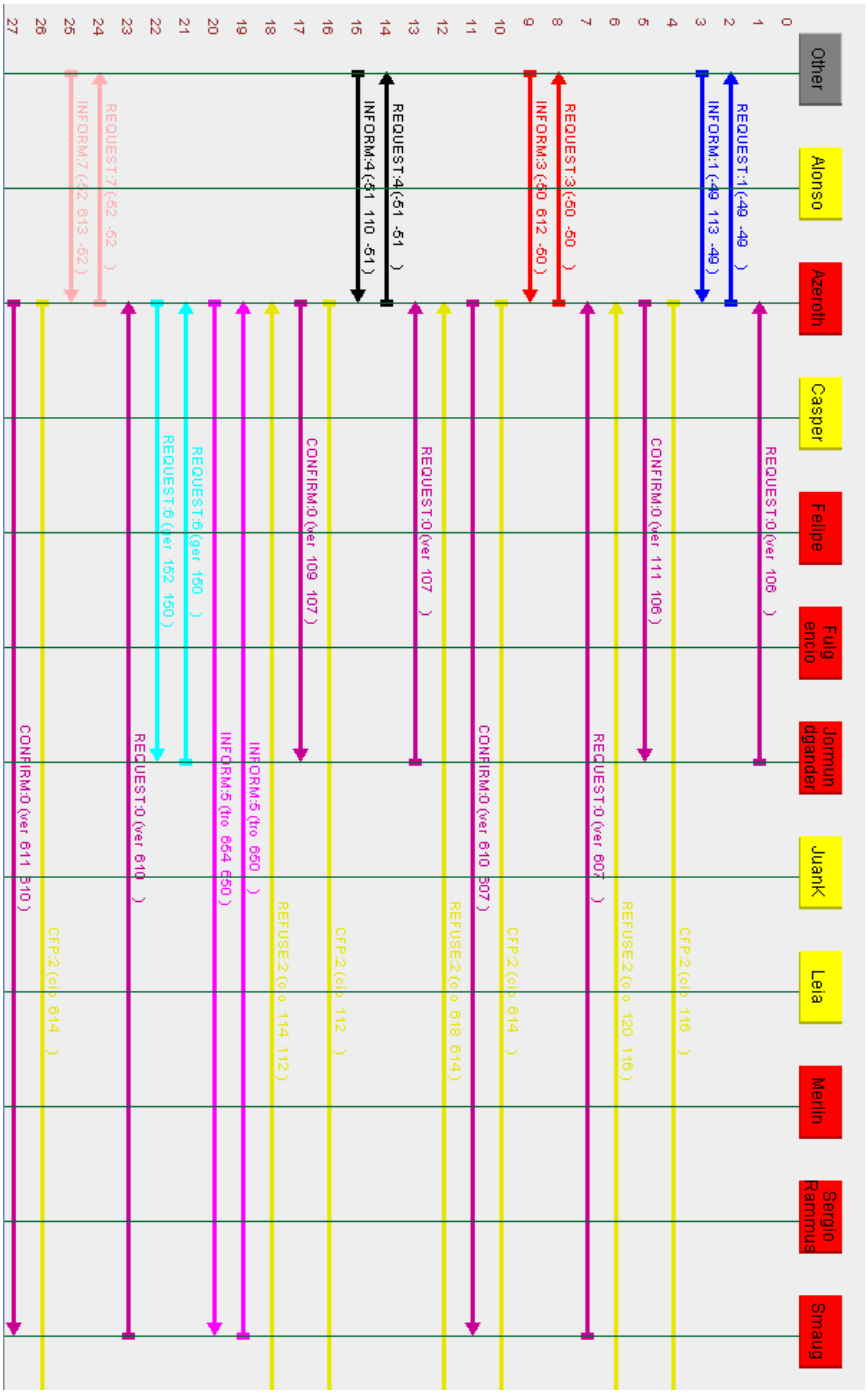


Figura A.2: Diagrama de ejemplo de intercambio de mensajes 2

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

Aeronautiques, C., Howe, A., Knoblock, C., McDermott, I. D., Ram, A., Veloso, M., Weld, D., SRI, D. W., Barrett, A., Christianson, D. et al. Pddl?the planning domain definition language version 1.2. *AIPS-98 Planning Competition Committee*, 1998.

Bellifemine, F., Caire, G., Poggi, A., Rimassa, G. y Jade, A. {A White Paper}. *Telecom Italia EXP magazine Vol*, vol. 3, 2008.

Bellifemine, F. L., Caire, G. y Greenwood, D. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007. ISBN 0470057475.

Bringsjord, S. y Ferrucci, D. *Artificial Intelligence and Literary Creativity: Inside the Mind of Brutus, a Storytelling Machine*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1999. ISBN 0805819878.

Coles, A., Fox, M., Halsey, K., Long, D. y Smith, A. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, vol. 173(1), páginas 1–44, 2009.

Coles, A. I., Fox, M., Long, D. y Smith, A. Teaching forward-chaining planning with javaff. En *Department of Computer and Information Sciences, University of Strathclyde, Glasgow, G1 1XH, UK*. 2008.

Dehn, N. Story generation after tale-spin. En *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'81, páginas 16–18. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1981.

FIPA. Foundation for intelligent physical agents fipa 97 specification part 2 agent communication language. 1997. Disponible en `http://www.fipa.org/specs/fipa00018/OC00018.pdf` (último acceso, Mayo, 2016).

FIPA. Fipa acl message structure specification. 2002. Disponible en `http://www.fipa.org/specs/fipa00061/SC00061G.pdf` (último acceso, Diciembre, 2002).

FIPA. Foundation for intelligent physical agents. 2016. Disponible en `http://www.fipa.org/` (último acceso, Mayo, 2016).

GÓMEZ-GAUCHÍA, H. y PEINADO, F. Automatic customization of non-player characters using players temperament. En *3rd International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE)*, vol. 4326, páginas 241–252. Springer, Springer, Darmstadt, Germany, 2006. ISBN ISBN-10 3-540-49934-2. I.

HOFFMANN, J. Ff: The fast-forward planning system. *AI magazine*, vol. 22(3), página 57, 2001.

HOFFMANN, J. Everything you always wanted to know about planning (but were afraid to ask). En *Proceedings of the 34th Annual German Conference on Artificial Intelligence (KI'11)* (editado por S. Edelkamp). spring, Berlin, Germany, 2011.

HOFFMANN, J. y NEBEL, B. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, páginas 253–302, 2001.

KLEIN, S. *Automatic Novel Writing: A Status Report*. Technical report. University of Wisconsin-Madison Department of Computer Sciences, 1973.

LACLAUSTRA, I. M., LEDESMA, J. L., MÉNDEZ, G. y GERVÁS, P. Kill the dragon and rescue the princess: Designing a plan-based multi-agent story generator. En *5th International Conference on Computational Creativity, ICCC 2014 (Late Breaking paper)*. Ljubljana, Slovenia, 2014.

LEBOWITZ, M. Story-telling as planning and learning. *Poetics*, vol. 14(6), páginas 483–502, 1985.

MAS, A. *Agentes software y sistemas multiagente : conceptos, arquitecturas y aplicaciones*. Prentice Hall, 2005. ISBN 8420543675.

MEEHAN, J. R. Tale-spin, an interactive program that writes stories. En *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'77, páginas 91–98. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1977.

PÉREZ Y PÉREZ, R. *MEXICA: A Computer Model of Creativity in Writing*. Ph.d. thesis, The University of Sussex, 1999.

THEUNE, M., FAAS, S., NIJHOLT, A. y HEYLEN, D. The virtual storyteller: story creation by intelligent agents. En *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, páginas 204–215. Springer, 2003.

TURNER, S. R. *MINSTREL: A computer model of creativity and storytelling*. Ph.d. thesis, University of California at Los Angeles, Los Angeles, CA, USA, 1992.

WOOLDRIDGE, M. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edición, 2009. ISBN 0470519460, 9780470519462.